

VOLUME VG003

CRASH COURSE IN
G7000 GAMES PROGRAMMING

BY GRAHAM THOMASSON

VOLUME VG003

CRASH COURSE IN G7000 GAMES PROGRAMMING

CHAPTER II PRODUCT FUNCTIONAL SPECIFICATIONS

Contents

✓	Section	1	Introduction
✓		2	The 8048 chip
✓		3	Displaying static objects
✓		4	Movement of objects and special effects
✓		5	The Game-Players' Peripherals
✓		6	Sound Effects
✓		7	Typical Program Structure
		8	Executive Routines
		9	Standard Techniques

✓	Appendix	1	References
✓		2	Detailed Memory Maps
✓		3	Executive Listing
✓		4	Timing of TV frame scan
✓		5	Library Routines and Development Utilities
✓		6	LSS-Tables

1 INTRODUCTION

1.1 A Game Player's View

The G7000 Games equipment adapts to a domestic ^(colour) TV set and consists of:

- The Video-pac computer, housed in a full alphanumeric keyboard unit.
- Two joystick pads with an action button. The joysticks can be moved in 8 distinguishable directions and are self-centering.
- A set of plug-in games cartridges, which are in fact ROMs.

Typical applications of the keyboard are for entering the players' names and for number and word games. It can also be used as a special input medium, such as in the music package.

The joysticks are particularly suited to the control of moving or rotating objects. The action button is used in dogfight-type games for firing missiles and in logic type games for making a move.

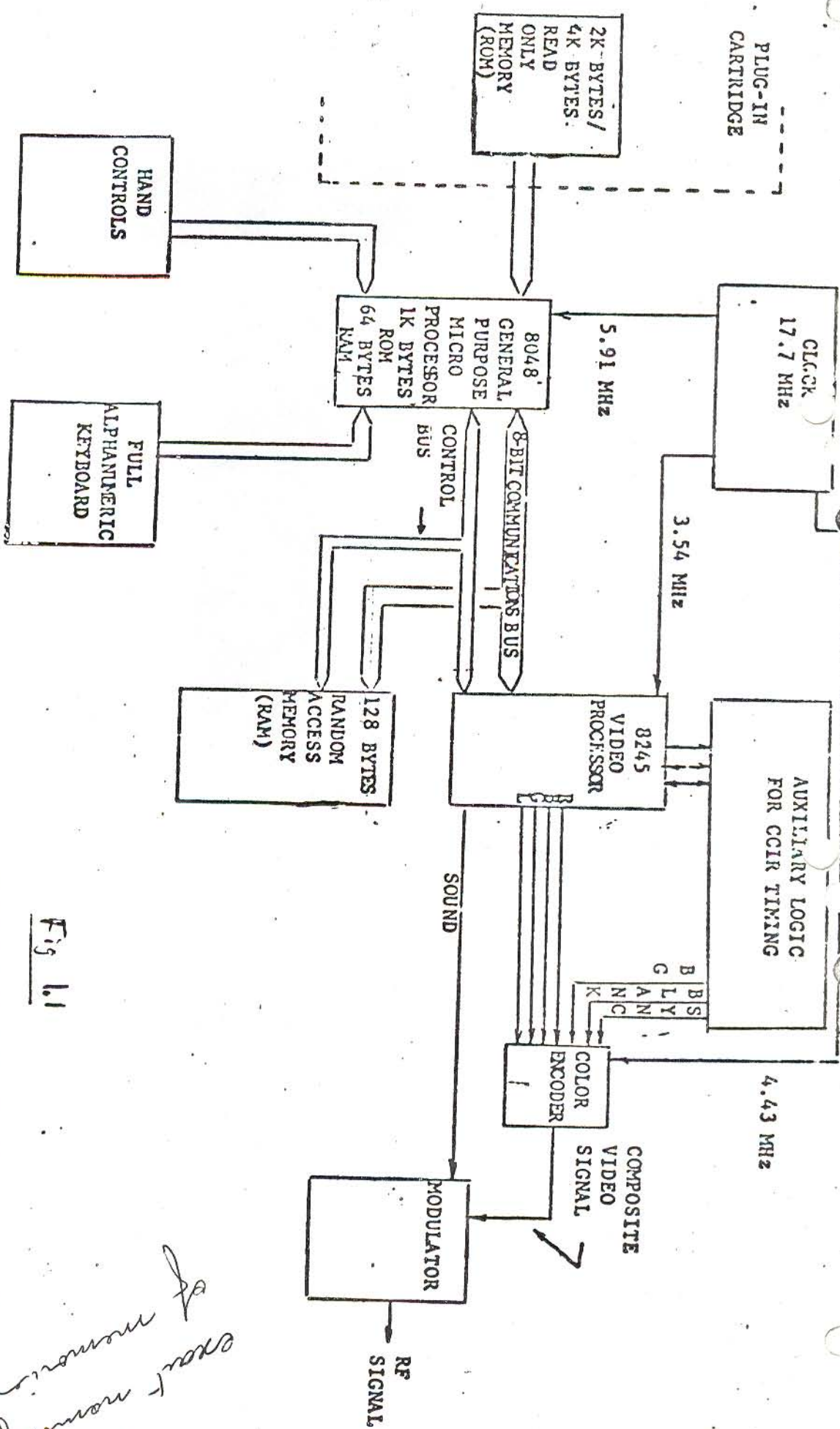
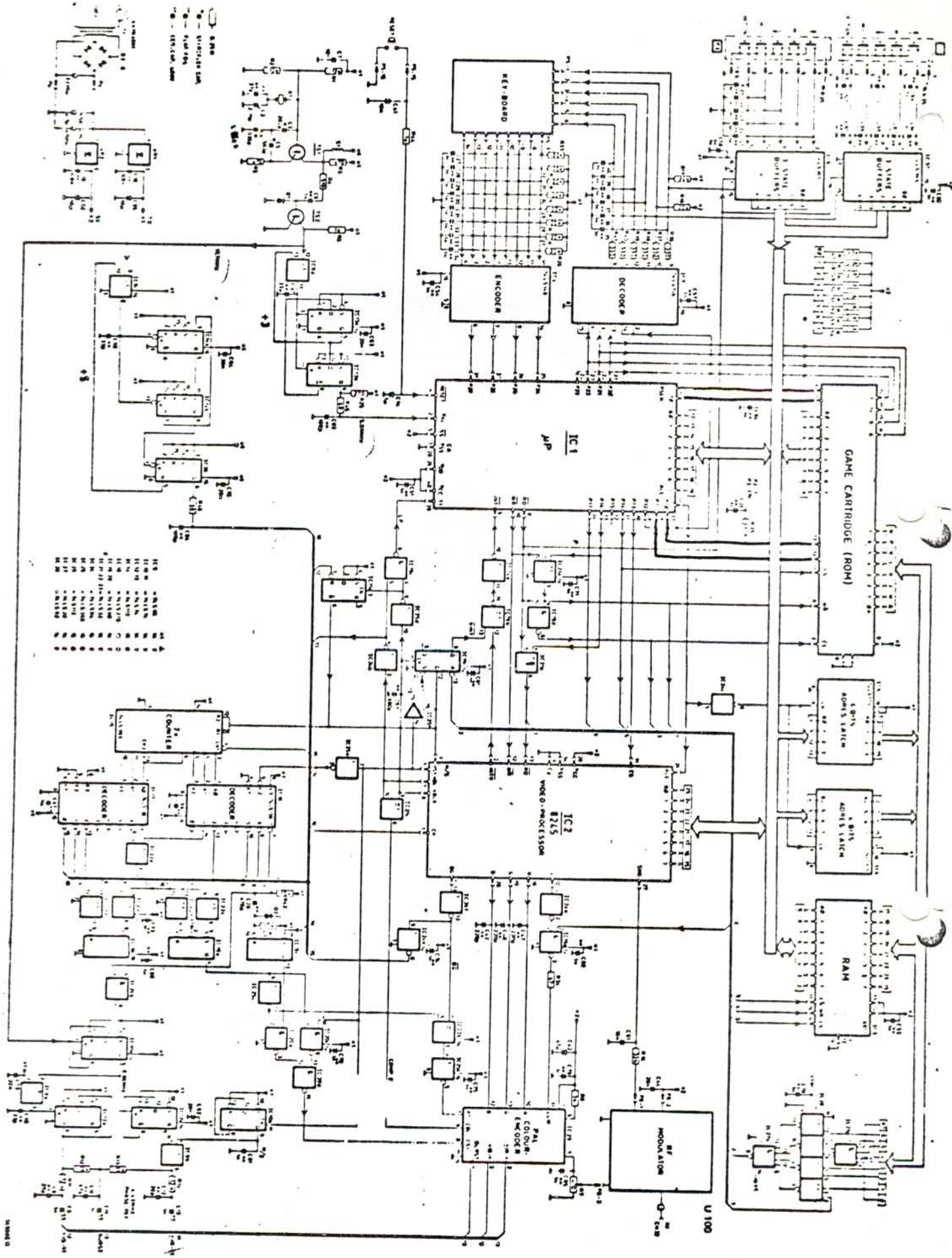


Fig 1.1
BLOCK DIAGRAM OF G7000 GAMES SYSTEM
(PAL VERSION)

exact naming of memory



1.2 Description of the G7000 components

Figure 1.1 shows a block diagram of the G7000 system.

The system basically consists of two ^{interconnected} microprocessors, video and sound modulators, various memories (including the plug in games ROM) and interfaces to the game controls and key board.

The Intel 8245 microprocessor deals with the construction of the video image. The chip contains ^{on-board} ROM and RAM (256 bytes). The RAM is also accessible to the other microprocessor and contains definitions of the ^{programmed} picture components. The ROM contains a program which runs continuously, reading out the RAM and using the information to construct the video image. This ROM-based program is of no real concern to the games programmer. It also contains definitions of a set of library or "major" objects, which are at the disposal of the games programmer.

The Intel 8048 microprocessor is the one which runs the games program. The chip contains a 1K ROM with executive routines and 64 bytes of RAM. An additional 128 bytes of external RAM are available as data store. The games program resides in the plug-in ROM cartridge. The hand controls and alphanumeric keyboard interface to the ports on this microprocessor. The 8048 has timer/counter functions, which are needed by the programmer to set up display data at the right time with respect to a TV frame scan cycle.

1.3 How a TV picture is constructed.

Although this topic may at first seem to be remote from the games programmer's area of activity, it is in fact important in connection with timing the enabling of the display and with timing the moment at which the graphics RAM is written to.

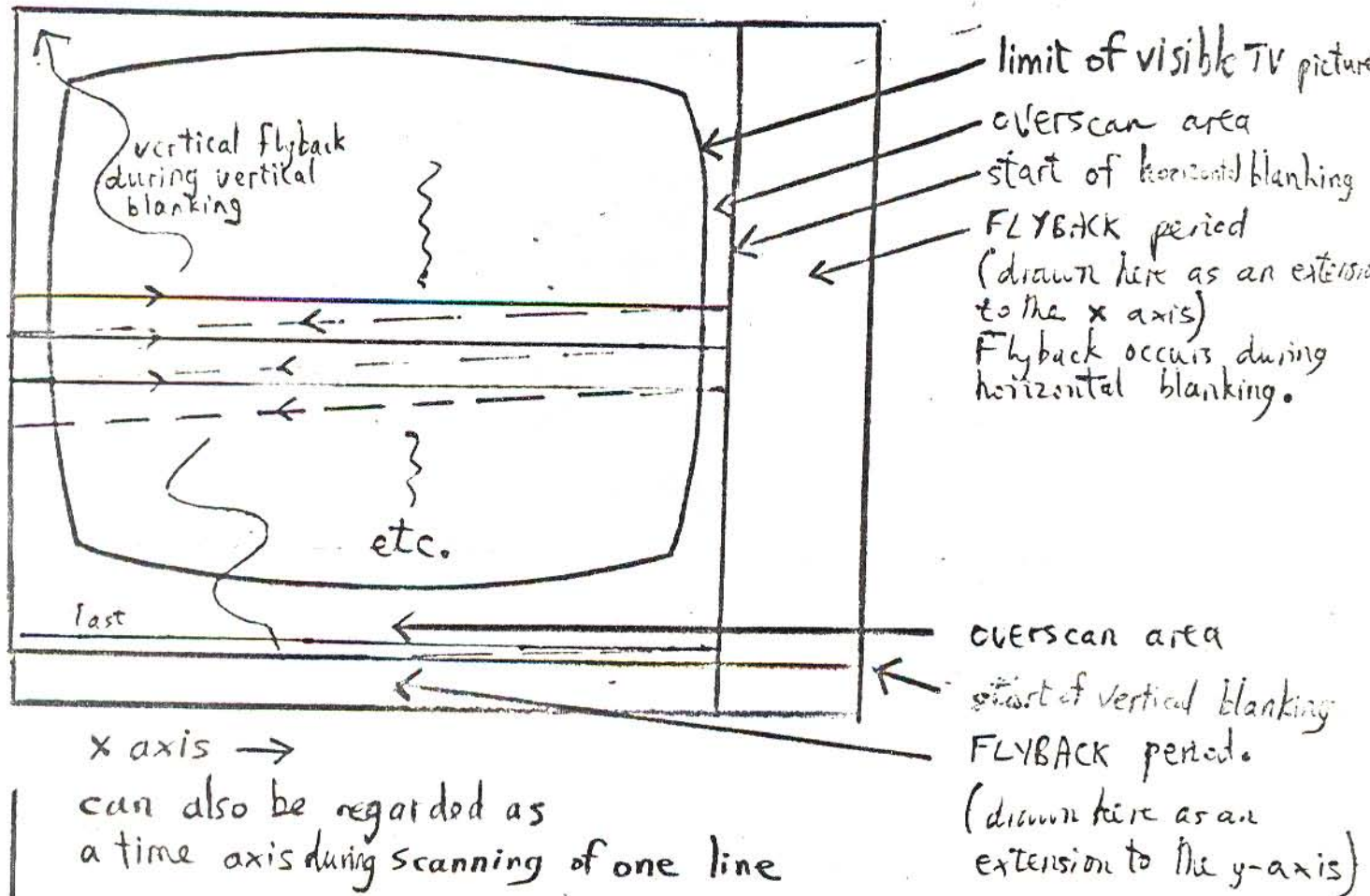
It also provides insight into the co-ordinate system. Furthermore, when predicting the effects of certain programming techniques on systems other than PAL, it is necessary to understand the physical consequences of the timing and graphics parameters employed.

A TV picture is produced by displaying a series of frames in rapid succession. Each frame is produced by scanning line for line, starting at the top left hand corner and working downwards. The lines are scanned during the sweep from left to right. The scanning process itself consists of modulating an electron beam so as to illuminate sensitive spots on the screen^(or not) as required in the construction of the picture. The electron beam is cut off during fly-back to the next line. This is called horizontal blanking^{or line blanking}. Shortly before all the lines have been written the beam is cut off again to return to the top left of the screen. This is called vertical blanking or field blanking. The line writing is then resynchronised and the next frame is written. Sometimes an "interlace factor" is used.

In this case a complete vertical scan of the TV tube is called a field, and successive fields interleave each other to produce a frame. The most common interlace factor is 2 so that alternate fields cover odd and even lines respectively.

The blanking periods ensure that there is no stray illumination during the Fly-backs. In fact the blanking starts somewhere just off screen so that the TV picture appears full. The active scanning from the edge of the TV screen to blanking is called "overscan".

The above is summarised diagrammatically below



x axis →

can also be regarded as
a time axis during scanning of one line

↓ y axis

can also be regarded as
a time axis during scanning of a field

When programming a game it is often important to know where the electron beam has got to in the scanning process.

Some actual figures are taken from the PAL system to illustrate this.

The start of a field scan is taken as the origin. The game's program knows when this occurs by virtue of an interrupt. For PAL, the line frequency is 15625 Hz which corresponds to 64 μ sec per line. The nominal number of lines is $312\frac{1}{2}$ per field, and 625 per frame (the interlace factor is 2).

Due to overscan, the visible part of a line cycle is less than 64 μ sec and the number of visible lines per field is less than $312\frac{1}{2}$.

Overscan is slightly dependent on the TV set and is less than the flyback times. Let us ignore overscan in what follows.

Horizontal blanking lasts for approx. 12 μ sec. This means that the active scan per line only lasts 52 μ sec. The 12 μ sec blanking interval may be useful to the programmer as it enables him to make small changes to the Graphics RAM without causing any undesired effects in the picture. The horizontal blanking can be tested for ~~and if required made to cause an~~ error.

Vertical blanking lasts approx. 1612 μ sec and is much more useful to the programmer. Note that this corresponds to more than 25 lines. The figure $312\frac{1}{2}$ should be thought of as a factor relating line frequency to field frequency and nothing more. The actual number of lines is seen to be less than 288, and then one must allow for overscan, reducing this even more.

Vertical blanking is always a good time to disable the display and alter the Graphics RAM. It is also possible to disable the display at pre-calculated intervals in the frame scan and make alterations then. By setting a timer interrupt at $64 \times n$ μ sec after frame synchronisation one can disable the display at line number n . This is discussed further in section 4. N.B. It is also possible to count lines using the counter feature of the 8048, and get an interrupt after n lines.

Finally, a note about colours.

Part of the electron beam modulation determines whether the beam is energized as it passes the sensitive spots on the screen which emit red green or blue light. From these three primary colours, other colours can be synthesized by combination. The G7000 allows only either full or no contribution to a particular colour. This gives the programmer the following colour control

RED

BLUE

GREEN

TURQUOISE = BLUE + GREEN

YELLOW = RED + GREEN

PURPLE = BLUE + RED

WHITE = RED + BLUE + GREEN

BLACK = no colour component.

2. The 8048 chip

The 8048 microprocessor is fully described Ref. 4. This section summarizes ^{where applicable in the context of the 8048} the main features. A concise table of the instruction set and also an instruction map are included.

Overview of the Instruction Set

The microprocessor works with an 8-bit data bus and an effective 12 bit address bus. There are very few instructions which work directly with more than 8 bits of the address bus. This means that the programmer is to some extent burdened with page-managing (1 page = 256 bytes) and memory bank control. The fact that the program is always in ROM accounts for the design of the instruction set — characteristics are very good ^{for instructions and} indexed addressing for the tiny on-board RAM and external RAM but relatively poor capabilities with ROM. Instructions ^{require} 1 or 2 cycles (of 2.5 μ s) to execute.

Memory Addressing

Refer to Appendix 2 for the memory organisation of all memory accessible to the 8048 chip. The different memory types to be considered are Internal (on-board) RAM, External RAM and ROM. The Graphics RAM is an alternative External RAM,

and is addressed in the same way as the External Data RAM. Switching between the two is accomplished by an I/O sequence described in Section 3.

The internal RAM consists of registers, stack space and free workspace. The layout is shown in Appendix 2.

The reader can refer to Reference 4 to see all the different instructions which work with this RAM area either as registers or as memory locations.

The External RAM uses the same addresses as the Internal RAM and therefore has completely separate instructions. It is not possible to do arithmetic and logical operations on the External RAM directly.

The ROM shares again the same addresses and works with its own instruction repertoire.

The diagram of the MOV instruction shows how data can be moved among the various memories.

There can be up to 9K ROM in the G7000 system. ~~and ROM 2128~~

The following paragraphs describe the functions in the internal RAM and the addressing of the ROM in more detail.

As can be seen from the diagram in Appendix 2, the Internal RAM consists of 2 sets of registers, a stack area and a user available area. The sets of registers are basically independent but conventions will dictate when one set is used and when the other (e.g. Interrupt routines use their own set, bank 0). Registers RD and RI are more powerful than the remainder since they can act as index registers. The stack is used by subroutine calls and interrupt processing. The stack pointer is part of a separate register, the PSW. This pointer takes on values from 000 (binary) to 111 to denote which stack cell is next to be used. Underflow and overflow cause wrapround. Each cell can store a 12 bit address and 4 bits of additional PSW information.

- ROM addressing becomes more difficult in the following progression: within the page — within a 2K bank — within a 4K field — outside 4K. There are two functions to consider: loading data from ROM, and transfer of control (jump instructions). Data can be loaded as immediate data and placed directly in RAM, register or the accumulator. Indexed ROM data is only possible where the data is present in the current page, or page 3. The page3 facility is used by the executive in the 67000 system and is not available to the games program itself. It is simply not possible to load data from a given 11- or 12-bit address.

- The situation regarding transfer of control is better and is summarized in the diagram on "Jump Instructions".

- On-page jump instructions are: an indirect jump (address in the accumulator) and a multiplicity of conditional jumps. There is an off-page unconditional jump which takes an 11 bit address as argument and will add the 12th bit (memory bank) if a SEL MB instruction has preceded it. The return instructions RET/RETR work with a full 12 bit address and can thus transfer control to anywhere in a 4K field, but

- To jump to a location outside a 4K field, the new field must be selected by outputting to bits 1 and 0 of port P1.

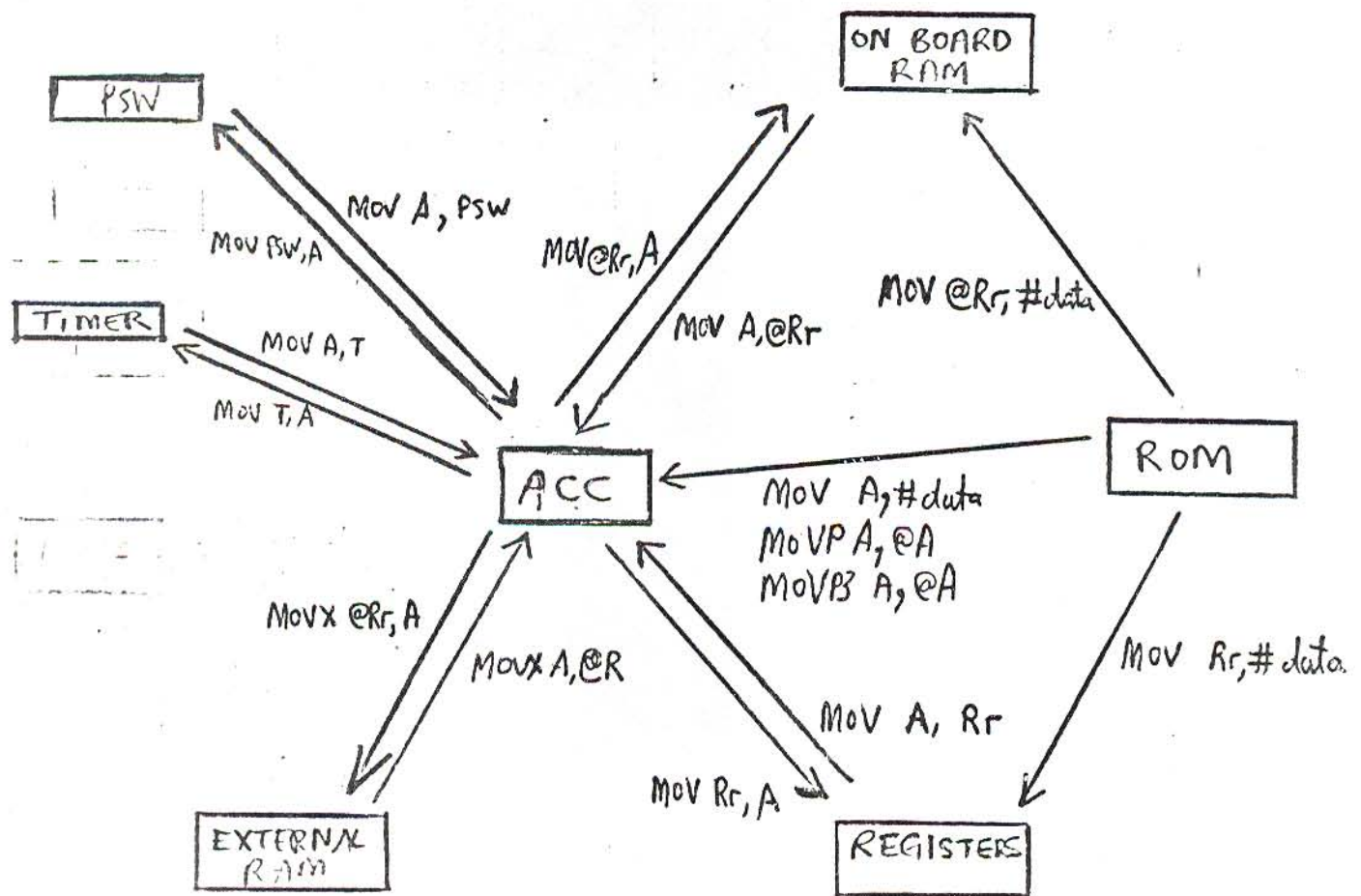
The codes are

11	Field 0
10	Field 1
01	Field 2
00	Field 3

- Any ^{executive} one field contains the 1K/ROM mapped in plus another 1K of games ROM in bank 0 plus another 1K of games ROM in bank 1. The remaining 1K of address space of bank 1 of a field is not used. (There may be exceptions in older existing games).

making
After the field selection via port 11 the processor needs a few microseconds to effect the field switch. Processing ^{then} continues in the new field, with no change to address bits 0-11. A control routine will be introduced into the library to remove the complexity from the programmer's code.

MOV- Instruction

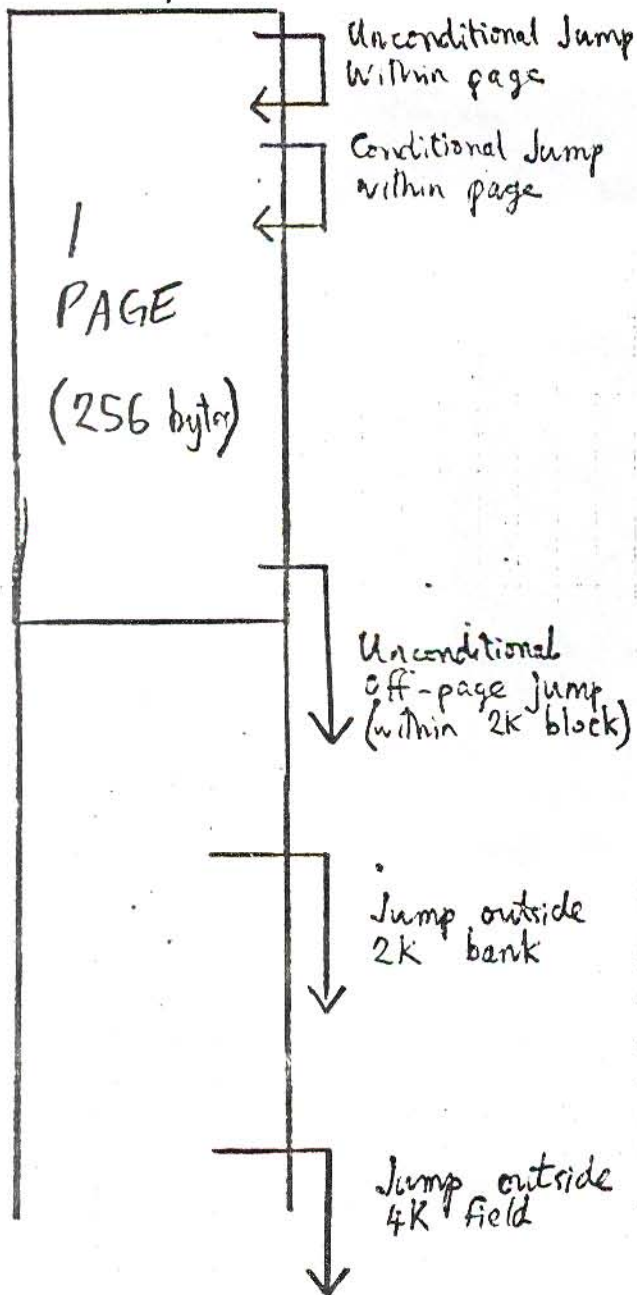


Notes 1. Transfer to and from ports 1 and 2 takes place by the IN and OUTL instructions.

2. There are many other instructions which effect data transfer
e.g. XCH exchange instructions
and arithmetic and logic instructions

3. The "REGISTERS" can also be addressed as "ON BOARD RAM"

Jump Instructions



• JMPP @A Indirect jump
(see also JMP below)

Jb	if acc bit b set
JC	if carry set
JFO	if flag 0 set
JFI	if flag 1 set
JNC	if no. carry
JNI	if interrupt input is low
JNT0	if Test 0 is low
JNT1	if Test 1 is low
JNZ	if Acc nonzero
JTF	if Timer Flag set
JTO	if Test 0 is high
JTI	if Test 1 is high
JZ	if Acc is zero
DJNZ	decrement & jump on nonzero

JMP
CALL
(see also RET/RETR below)

JEL MBI select memory bank
jump or call

RET }
RETR }

(Requires I/O instructions
via port 1 to select
the Rom. Discussed in
the text).

8048/8049 INSTRUCTION SET SUMMARY

Mnemonic	Description	Bytes	Cycle	Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A, R	1	1	Subroutine	CALL	2	2
	ADD A, @R	1	1		RET	1	2
	ADD A, #data	2	2		RETR	1	2
	ADDC A, R	1	1	Flags	CLR C	1	1
	ADDC A, @R	1	1		CPL C	1	1
	ADDC A, #data	2	2		CLR F0	1	1
	ANL A, R	1	1		CPL F0	1	1
	ANL A, @R	1	1		CLR F1	1	1
	ANL A, #data	2	2		CPL F1	1	1
	ORL A, R	1	1	Data Moves	MOV A, R	1	1
	ORL A, @R	1	1		MOV A, @R	1	1
	ORL A, #data	2	2		MOV A, #data	2	2
	XRL A, R	1	1		MOV R, A	1	1
	XRL A, @R	1	1		MOV @R, A	1	1
	XRL A, #data	2	2		MOV R, #data	2	2
	INCA	1	1		MOV @R, #data	2	2
	DECA	1	1		MOV A, PSW	1	1
	CLR A	1	1		MOV PSW, A	1	1
	CPL A	1	1		XCH A, R	1	1
	DA A	1	1		XCHA, @R	1	1
	SWAP A	1	1		XCHD A, @R	1	1
Input/Output	RLA	1	1		MOVX A, @R	1	2
	RLC A	1	1		MOVX @R, A	1	2
	RR A	1	1		MOVP A, @A	1	2
	RRC A	1	1		MOVP3 A, @A	1	2
	IN A, P	1	2	Timer/Counter	MOV A, T	1	1
	OUTL P, A	1	2		MOV T, A	1	1
	ANL P, #data	2	2		STRT T	1	1
	ORL P, #data	2	2		STRT CNT	1	1
	INS A, BUS	1	2		STOP TCNT	1	1
	OUTL BUS, A	1	2		EN TCNTI	1	1
	ANL BUS, #data	2	2		DIS TCNTI	1	1
Registers	ORL BUS, #data	2	2	Control	EN I	1	1
	MOVD A, P	1	2		DIS I	1	1
	MOVD P, A	1	2		SEL RB0	1	1
	ANLD P, A	1	2		SEL RB1	1	1
	ORLD P, A	1	2		SEL MB0	1	1
Branch	INC R	1	1		SEL MB1	1	1
	INC @R	1	1		ENT0 CLK	1	1
	DEC R	1	1		NOP	1	1
	JMP addr	2	2	Branch			
	JMPP @A	1	2				
	DJNZ R, addr	2	2				
	JC addr	2	2				
	JNC addr	2	2				
	JZ addr	2	2				
	JNZ addr	2	2				
	JT0 addr	2	2				
	JNT0 addr	2	2				
	JT1 addr	2	2				
	JNT1 addr	2	2				
	JF0 addr	2	2				
	JF1 addr	2	2				
	JTF addr	2	2				
	JNI addr	2	2				
	JBb addr	2	2				

8048 Instruction map

		second hexadecimal digit of OP-code															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
first hexadecimal digit of OP-code	0	NOP		OUTL BUSA	ADD A, #	JMP ₀	ENI		DECA	INS ABUS	IN A, Pp			MOVD A, Pp			
	1	INC @Rr	JB0	ADDC A, #	CALL	DIS I	JTF addr	INCA	INC Rr ₀₋₇								
	2	XCHA @Rr		MOV A, #	JMP ₁	ENTCNTI	JNT0 addr	CLRA	XCH Rr ₀₋₇								
	3	XCHDA @Rr	JB1		CALL	DIS TCNTI	JTO addr	CPLA					MOVD Pp, A				
	4	ORLA @Rr	MOV A, T	ORL A, #	JMP ₂	STRT CNT	JNT1 addr	SWAP A	ORL A, Rr ₀₋₇								
	5	ANLA @Rr	JB2	ANL A, #	CALL	STRT T	JT1 addr	DAA	ANL A, Rr ₀₋₇								
	6	ADDA @Rr	MOV T, A		JMP ₃	STOP TCNT		RRCA	ADD A, Rr ₀₋₇								
	7	ADDC A @Rr	JB3		CALL	ENTO CLK	JF1 addr	RRA	ADDC A, Rr ₀₋₇								
	8	MOVXA @Rr		RET	JMP ₄	CLRF0	JN1 addr		ORL BUS, #	ORL Pp, #			ORLD Pp, A				
	9	MOVX @Rr, A	JB4	RETR	CALL	CPL F0	JNZ addr	CLRC	ANL BUS, #	ANL Pp, #			ANLD Pp, A				
	A	MOV @Rr, A		MOV P, A @A	JMP ₅	CLRF1		CFLC	MOV Rr, A ₀₋₇								
	B	MOV @Rr, #	JB5	JMPF @A	CALL	CPL F1	JF0 addr		MOV Rr, # ₀₋₇								
	C					JMP ₆	SEL RBO	JZ addr	MOV A, PSW	DECRr ₀₋₇							
	D	XRLA @Rr	JB6	XRL A, #	CALL	SEL RB1		MOV PSWA	XRL A, Rr ₀₋₇								
	E				MOV P, BA @A	JMP ₇	SEL MBO	JNC addr	RLA	DJNZ Rr, addr							
	F	MOVA @Rr	JB7		CALL	SEL MB1	JC addr	RLCA	MOVA, Rr ₀₋₇								

: data (8 bits) @ : addressed by



1 byte / 1 cycle



1 byte / 2 cycles



2 bytes / 2 cycles



undefined

Reset and Interrupt Handling

When the reset button is pressed a reset pulse is sent to the 8048 which then

1. Sets program counter to zero.
2. Sets stack pointer to zero.
3. Selects register bank 0.
4. Selects memory bank 0.
5. Sets BUS to high impedance state. (except when EA = 5V)
6. Sets Ports 1 and 2 to input mode.
7. Disables interrupts (timer and external)
8. Stops timer.
9. Clears timer flag.
10. Clears F0 and F1.
11. Disables clock output from T0.

N.B. Counter content not affected

Processing continues at address 0. This is in the executive of the G7000 system and a jump is made to 400H. Each memory field must be capable of handling the reset function from this address. Obviously, only one field need do the real work and the others can transfer to it.

Interrupts from devices other than the timer cause an automatic CALL to address 3. The executive vectors this on to address 402H. The same comments apply concerning memory fields as above. An interrupt routine ends with a RETR which restores the environment that prevailed at the time of the interrupt.

Interrupts from the timer cause an automatic call to address 7. The executive vectors this on to address 404H. These interrupts do not take priority over other interrupts - an interrupt routine is never itself interrupted.

Other interrupts in the G7000 are

- Sound
- Vertical TV blanking

The joysticks, action buttons and keyboard are noninterrupting

I/O Ports

Ports 1 and 2 are used by the G7000, and also the BUS.

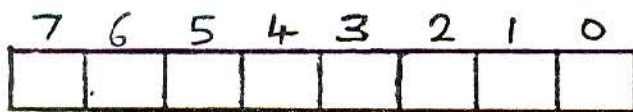
The diagrams on the following pages show the bit functions for each port.

For Port 1 the use of the ROM select bits ^{has been} explained earlier in this section. The bits for selecting the Graphics RAM and/or External RAM and the "One Way Transfer" bit are covered in Section 3. The Keyboard and hand control enable bit is self explanatory. The Force Video high bit pushes half intensity signals to full intensity.

Port 2 is devoted to the keyboard. The bits used define whether a key has been pressed and if so which one. An executive routine (KEYBRD) converts this input into standard keycodes.

Port Assignments

Port 1



Game ROM select

00 = 6-8K (field 3)

01 = 4-6K (field 2)

10 = 2-4K (field 1)

11 = 0-2K (field 0)

Keyboard and hand control enable

0 = enable 1 = disable

Select graphics RAM

0 = Active

Select external RAM

0 = Active

Hand control input enable

0 = Active

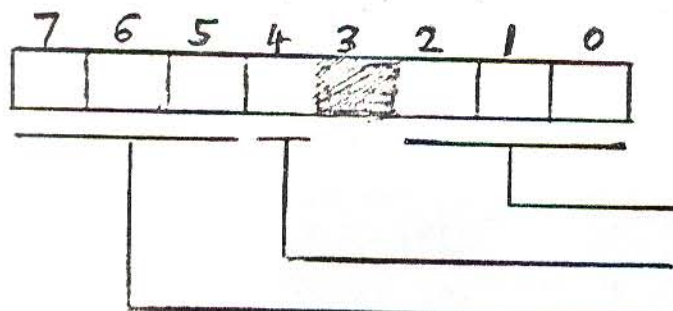
One way transfer (Read External RAM,
Write to Graphics RAM) 1 = Active

Force video high

0 = Active

Port 2

Keyboard Input + Output select



Keyboard Row Select

1 = Active

Valid Key Data

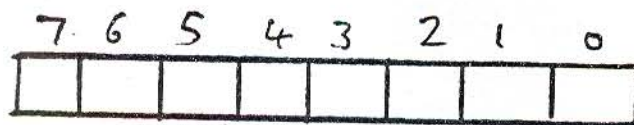
0 = Active

Keyboard Column Data

0 = Active

Keyboard is 6 rows x 8 columns

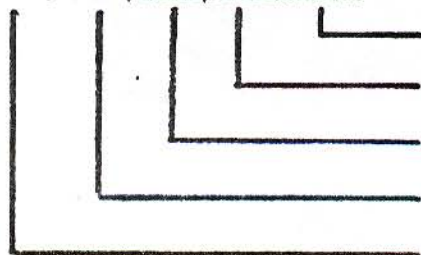
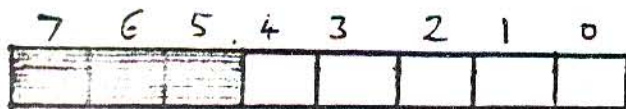
Hand Control Select



Nr 0-7 select

BLIS

(Joystick and Action Button Input)



Up
Right
Down
Left
Fire Button

0 = Active
0 = Active
0 = Active
0 = Active
0 = Active

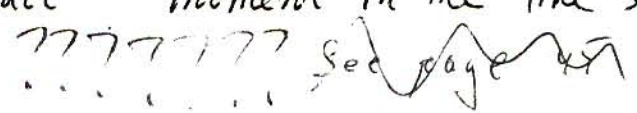
Use of Timers and Counters

There is one 8-bit register in the 8048 which can serve either as a timer or as an event counter. There is also a flag, TF, to indicate timer or event count overflow. The setting of this flag can be made to cause an interrupt.

The timer counts intervals of 32 instruction cycles. These intervals are 80 μ sec. The timer can be set to n so that after n intervals of 80 μ sec it will overflow. The delay range is 80 μ sec to 20 msec. The software can count overflow occurrences so as to provide for longer delays.

Note: The timer must be stopped at the moment of loading it.

When working without interrupts the JTF instruction can be used to test for overflow. This instruction clears the TF after execution.

The event counter ^{is similar to the timer but} counts video horizontal scan lines. These last 64 μ sec. ^{counts intervals} The exact moment in the line scan which causes the interrupt is  See page 47

This feature is very important in the G7000 system as it enables the programmer to make use of predefined lines in the picture for object movement. This is discussed in detail elsewhere.

Interrupt handling has been dealt with earlier in this section.

Summary of Timer/Counter instructions

MOV A,T Read Timer/Counter (Timer/Counter may be running)
MOV T,A Load Timer/Counter (Timer/Counter must be stopped)
STRT T Start Timer
STRT CNT Start Counter
STOP TCNT Stop Timer/Counter
EN TCNTI Enable Timer/Counter Interrupts
DIS TCNTI Disable Timer/Counter Interrupts

JTF

Jump if Timer Flag is set and clear this flag

3 DISPLAYING STATIC OBJECTS

A picture is built up from a limited number of components. These are: a background, a grid, library objects ("majors"), and user-described objects ("minors"). It is not possible to independently address points covering the whole screen, as with many home-computer systems.

The picture components are set up by writing to the Graphics RAM. Refer to Appendix 2 for a map of this memory chip and a summary of its functions. Once the Graphics RAM has been composed, the 8245 chip takes care of the actual display by reading it and interpreting it so as to generate the TV picture.

In order to write to the Graphics RAM it must be selected. This is necessary because the 8048 can only address one external RAM at a time (for a given I/O direction). Selecting the Graphics RAM may be combined with unselecting the External Data RAM. There are two modes in which the Graphics RAM can be addressed: one-way transfer mode and read/write mode. In the one-way transfer mode, the External Data RAM remains addressable for reading and the Graphics RAM is the object of write instructions. In read/write mode the Graphics RAM completely replaces the External Data RAM. The RAM selection operation involves an I/O instruction via port nr 1. The actual instructions to set up one-way transfer mode are:

```
BLKT1 EQU 07CH
BLKT2 EQU 0E7H
```

```
ORL PI, # BLKT1
```

```
ANL PI, # BLKT2
```

```
; activate one-way transfer
```

```
; and select both Ext RAM and 8245
```

This maps the bit pattern $\overset{\text{on port 1}}{\text{XXXXXXXX}}$ into X11001XX . The functions affected are

- one way transfer active
- hand control input disable

- Select External Data RAM.
- Select Graphics RAM.
- Disable keyboard.

25

To set up read/write mode on the Graphics RAM the instructions are

```
PTICLR EQU 0BCH
S8244 EQU 0B7H
ORL PI, #PTICLR
ANL PI, #S8244
```

But under high pushed mode

To restore read/write operation with the External Data RAM, the sequence is

```
PTICLR EQU 0BCH
ERAM EQU 0AFH
ORL PI, #PTICLR
ANL PI, #ERAM
```

Certain parts of the Graphics RAM can be read or written to without disabling the display. Before any object can be created, however, the display must be disabled. This inhibits the generation of all existing objects, but the background and grid are not affected.

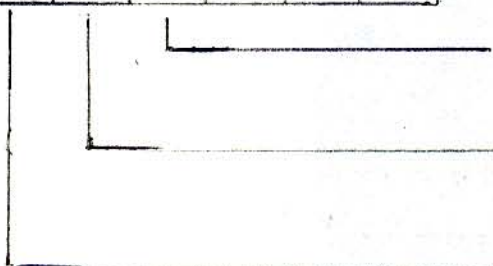
The display is disabled by clearing bit 5 in Graphics RAM address 0A0H, and enabled by resetting this bit.

The period during which the display is disabled must be carefully chosen. There should be no objects or parts of objects to display during this period.

This involves setting the timer so as to disable at the right part of the frame scan cycle. The section on Moving Objects discusses this in further detail. The rest of this section explains how to display the various picture components.

The Background

The only settings are the colour components. The relevant bits are in Graphics RAM address A3:



= 1 for BLUE component

= 1 for GREEN component

= 1 for RED component

BLACK = no colour component

WHITE = RED + GREEN + BLUE

YELLOW = RED + GREEN

TURQUOISE = BLUE + GREEN

PURPLE = RED + BLUE

It is possible to obtain a striped background effect by altering the background settings at the right part of the frame scan cycle. The display need not be disabled to do this. However, this requires due attention to timing and is further discussed in the next Section.

The Grid

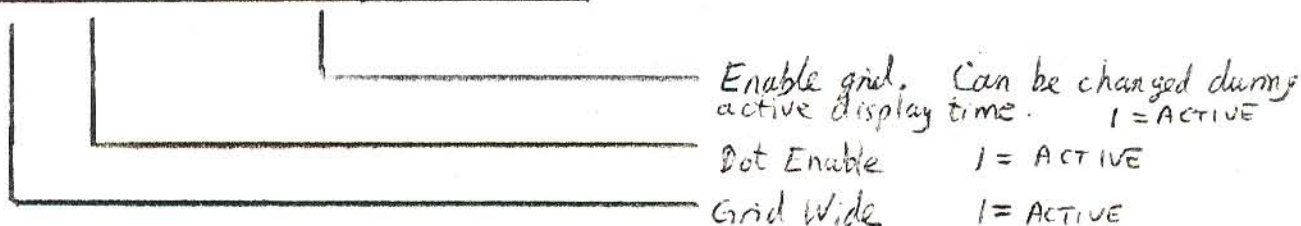
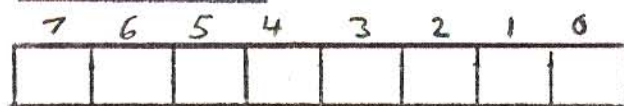
28

A grid, or selected segments of a grid, can be drawn on top of the background. For a rough idea of what the grid looks like and its positioning relative to the edge of the screen, refer to Fig. 3.1. The narrow tramlines represent the normal grid segments. It is also possible to select a wide grid, in which case the vertical segments are extended to the right as far as the next vertical segment. The rectangles on the far right of Fig. 3.2 indicate the wide grid extensions to the last column of vertical grid segments. Wide grid segments join up with their neighbours.

A further feature is to present a dot array located at all intersections of (narrow) grid segments, including the apparent intersections at the edge of the grid. The dot array has the same colour and intensity as the grid but is otherwise independent. The grid (and/or dot array) can be displayed at full intensity or half intensity.

Programming The Grid

Word A0



Word A3

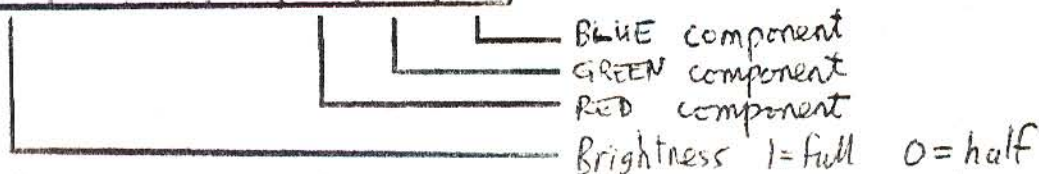
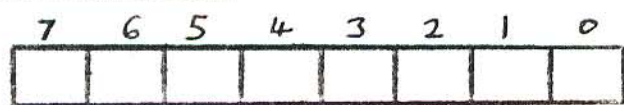


Fig 3.1 GRID ELEMENT ADDRESSING

	C0/0	C1/0	C2/0	C3/0	C4/0	C5/0	C6/0	C7/0	C8/0	
E0/0	E1/0	E2/0	E3/0	E4/0	E5/0	E6/0	E7/0	E8/0	E9/0	
C0/1	C1/1	C2/1	C3/1	C4/1	C5/1	C6/1	C7/1	C8/1		
E0/1	E1/1	E2/1	E3/1	E4/1	E5/1	E6/1	E7/1	E8/1	E9/1	
C0/2	C1/2	C2/2	C3/2	C4/2	C5/2	C6/2	C7/2	C8/2		
E0/2	E1/2	E2/2	E3/2	E4/2	E5/2	E6/2	E7/2	E8/2	E9/2	
C0/3	C1/3	C2/3	C3/3	C4/3	C5/3	C6/3	C7/3	C8/3		
E0/3	E1/3	E2/3	E3/3	E4/3	E5/3	E6/3	E7/3	E8/3	E9/3	
C0/4	C1/4	C2/4	C3/4	C4/4	C5/4	C6/4	C7/4	C8/4		
E0/4	E1/4	E2/4	E3/4	E4/4	E5/4	E6/4	E7/4	E8/4	E9/4	
C0/5	C1/5	C2/5	C3/5	C4/5	C5/5	C6/5	C7/5	C8/5		
E0/5	E1/5	E2/5	E3/5	E4/5	E5/5	E6/5	E7/5	E8/5	E9/5	
C0/6	C1/6	C2/6	C3/6	C4/6	C5/6	C6/6	C7/6	C8/6		
E0/6	E1/6	E2/6	E3/6	E4/6	E5/6	E6/6	E7/6	E8/6	E9/6	
C0/7	C1/7	C2/7	C3/7	C4/7	C5/7	C6/7	C7/7	C8/7		
E0/7	E1/7	E2/7	E3/7	E4/7	E5/7	E6/7	E7/7	E8/7	E9/7	
D0/0	D1/0	D2/0	D3/0	D4/0	D5/0	D6/0	D7/0	D8/0		

The above diagram indicates which bits of which bytes address which segments. The bytes in use are C0-C8, D0-D8 and E0-E9. One bit represents 1 segment (1=ACTIVE). The format is
 byteaddress/bitnumber.

Displaying Library Objects (MAJORS).

It is possible to display 12 independent library objects (SINGLE MAJORS) and 4 groups of 4 library objects (GROUP MAJORS).

The four members of a group major can be separate objects, in their own colour, but they will have the same Y-value (that of the fourth in the group) and the X-spacing will be 16. The X-coordinate of the first is taken from the X-value in the fourth of the group and the other three are spaced to the right from this.

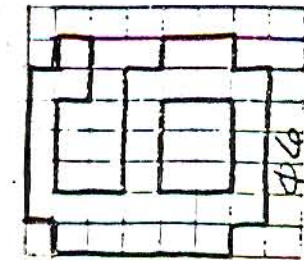
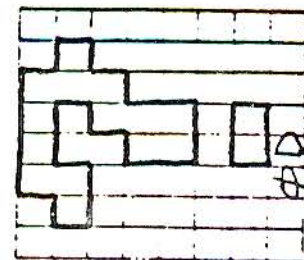
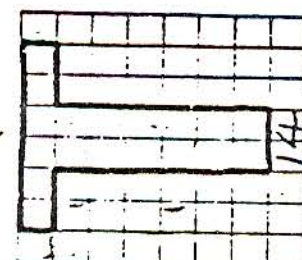
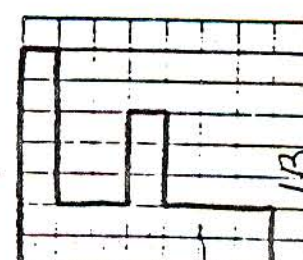
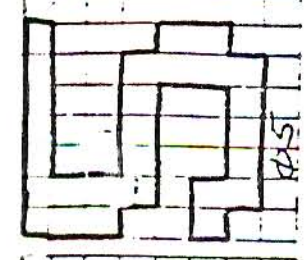
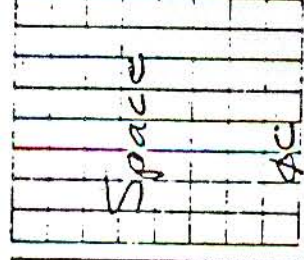
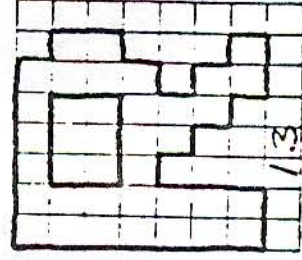
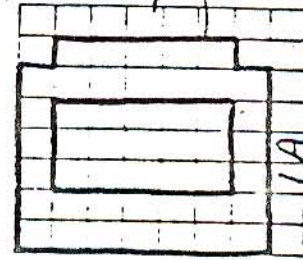
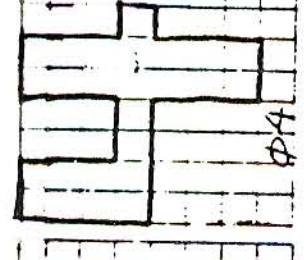
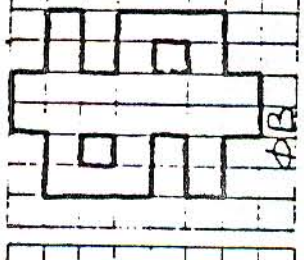
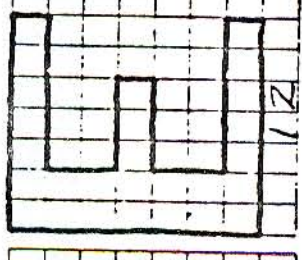
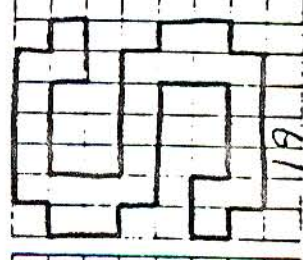
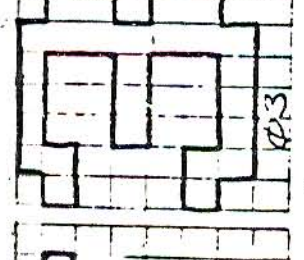
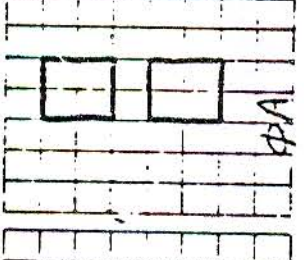
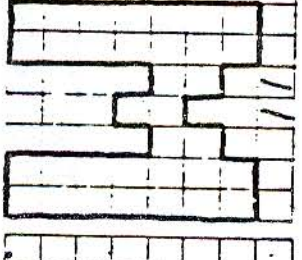
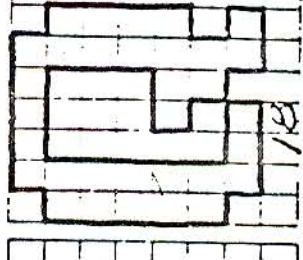
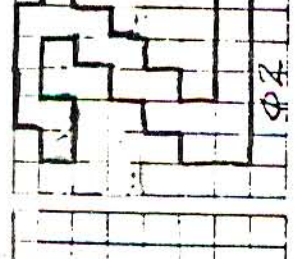
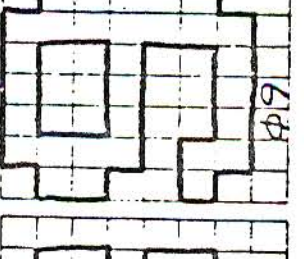
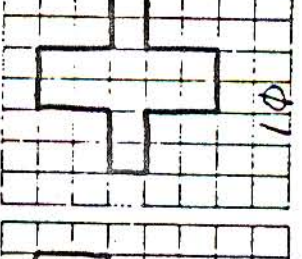
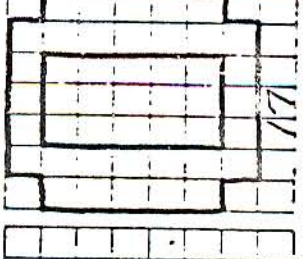
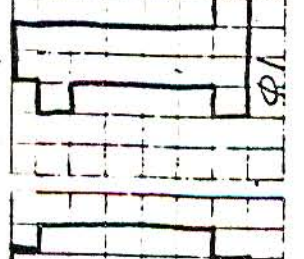
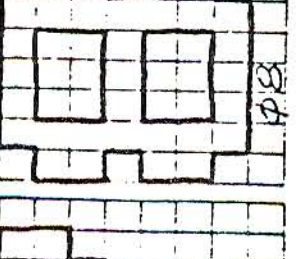
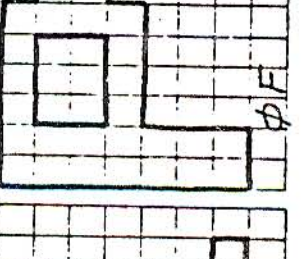
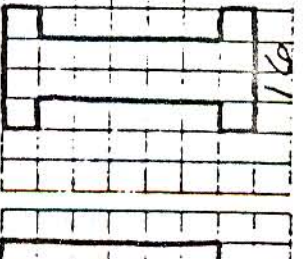
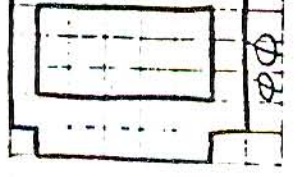
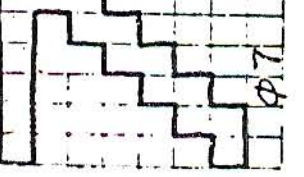
The library objects are composed on an 8×8 matrix. Column nr 8 is not used by the alphanumerics so as to act as spacing between the letters of words etc. Row nr 8 is not used at all in any library object. The 8 rows represent 16 Y-coordinate values, each row beginning on an even line number. The unit of an element in the 8×8 matrix is called a pixel and is 280 ns long and 2 lines high.

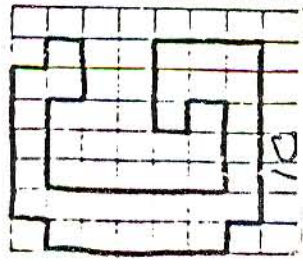
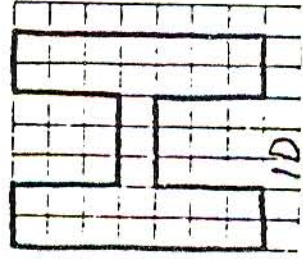
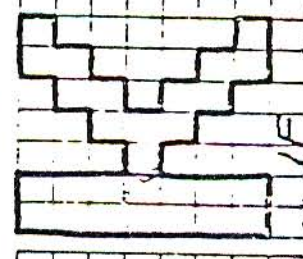
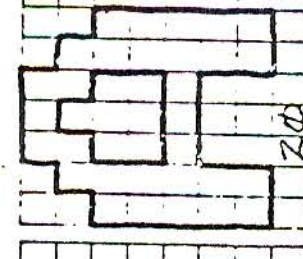
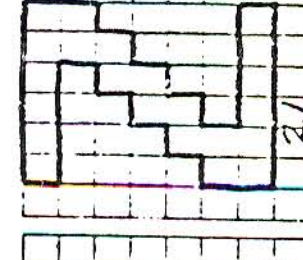
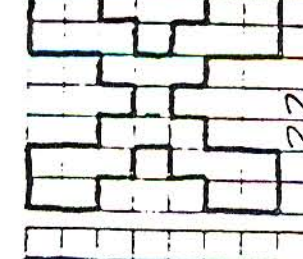
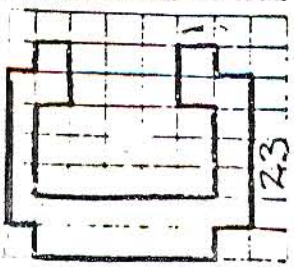
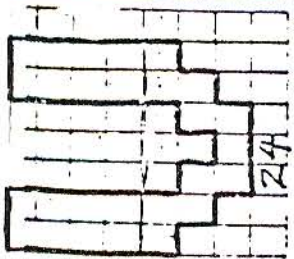
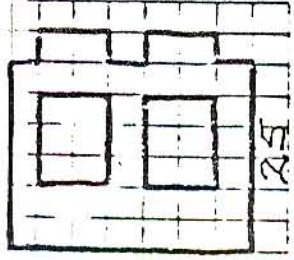
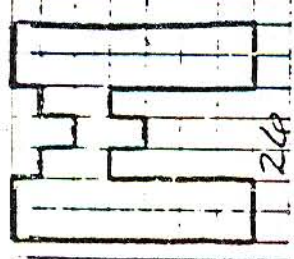
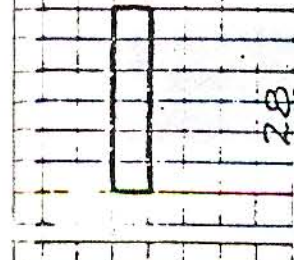
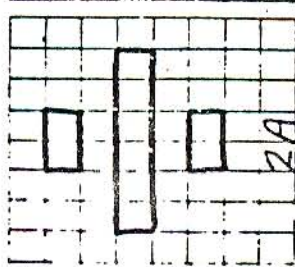
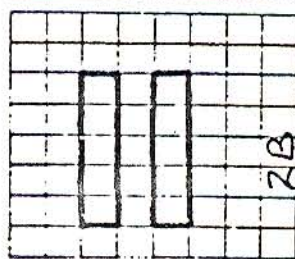
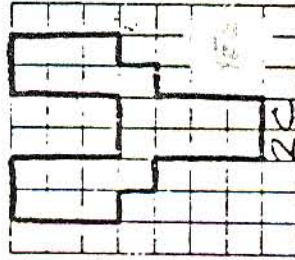
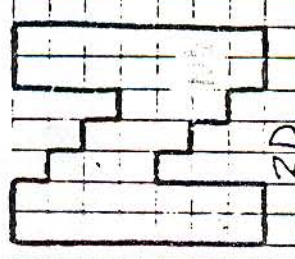
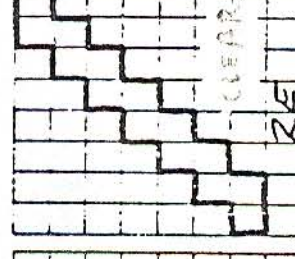
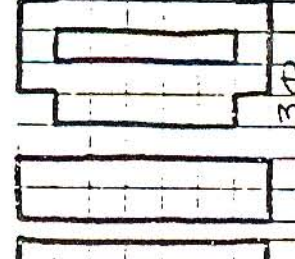
Majors are displayed on top of the grid and background. Not all colour combinations are satisfactory - refer to figure 3.3 for acceptable object colours on various backgrounds.

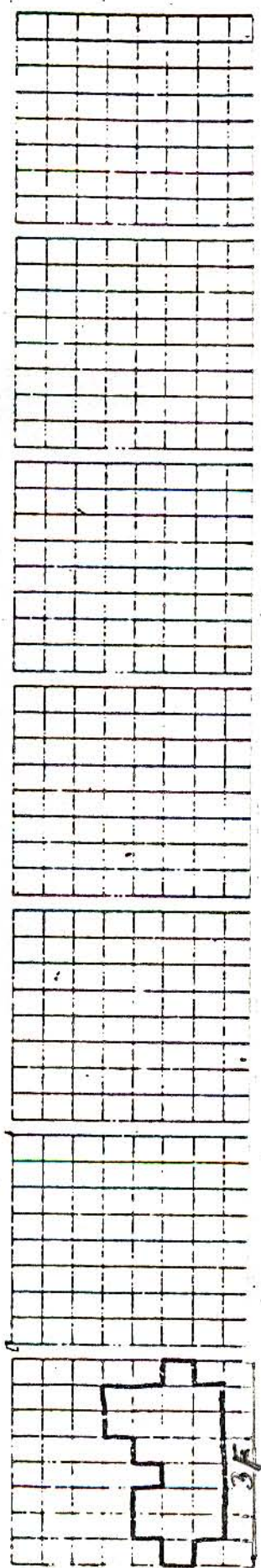
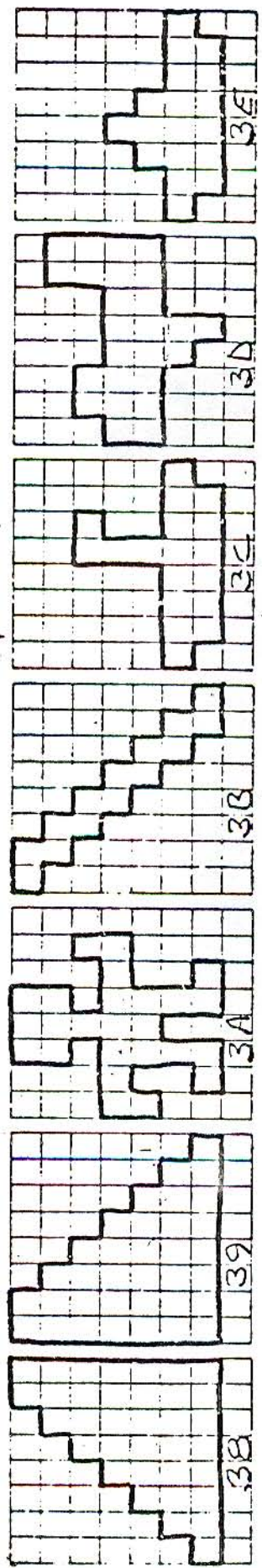
The following sheets show the set of objects and their codenumbers (keycodes).

TABLE OF LIBRARY OBJECTS (MAJORS)

31

			
$\Phi 6$	ΦD	$\Phi 14$	ΦB
			
$\Phi 5$	Space ΦC	$\Phi 13$	ΦA
			
$\Phi 4$	ΦB	$\Phi 12$	$\Phi 18$
			
$\Phi 3$	ΦA	$\Phi 11$	$\Phi 18$
			
$\Phi 2$	$\Phi 9$	$\Phi 10$	$\Phi 17$
			
$\Phi 1$	$\Phi 8$	ΦF	$\Phi 16$
			
$\Phi \Phi$	$\Phi 7$	ΦE	$\Phi 15$

							
1D	1E	1F	2A	2B	2C	2D	2E
							
23	24	25	26	27	28	29	3A
							
31	32	33	34	35	36	37	38



Programming MAJOR OBJECTS

For the addresses in the Graphics RAM of each Major, refer to the memory maps in Appendix 2.

Single Major object (the addresses ^{given} are those of the first)

Address	7	6	5	4	3	2	1	0
10	Y	Y	Y	Y	Y	Y	Y	0
11	X	X	X	X	X	X	X	X
12	LSS	LSS	LSS	LSS	LSS	LSS	LSS	LSS
13					B	G	R	LSS9

Y co-ordinate (must be even!)

X co-ordinate

LSS see below

B = Blue Component (1 = ACTIVE)

G = Green Component (1 = ACTIVE)

R = Red Component (1 = ACTIVE)

The LSS bits

LSS stands for Linear Select Store.

The LSS bits contain the keycode information, in a somewhat disguised form. The definition of the LSS value is

$$LSS = (8 \times \text{keycode}) - (Y/2)$$

As this may yield a negative result, there are 9 LSS bits. LSS9 is the most significant bit.

There is a subroutine in the executive for calculating LSS values from the keycode and Y value, LSSCAL.

Restrictions with Majors

- 1) Majors may not overlap other Majors, except for row 8 of the matrix.
- 2) Majors may not be clipped by use of a disable display.

Co-ordinate Range (also applies to minor objects)

This is
PAL

dependent on the TV system in use.

X in the range 0 to 167 to display an entire object.

X may go as far as 170 to still display the extreme left of an object.

X may also take on the value -227 to display at an effective address of -1.

Y in the range 0 to 227 to display an entire object.

Y may go as far as 242 to still display the top of an object. For major objects Y must be even.

NTSC X in the range 0 to 183
Y in the range 0 to 242

To delete a Major, the Y-coordinate can be set to 0F8H. Alternatively, the keycode for a blank (0C) can be used in the LSS calculation. This method also works for Group Majors.

Programming GROUP MAJOR OBJECTS

These are programmed in the same way as Single Major Objects except:

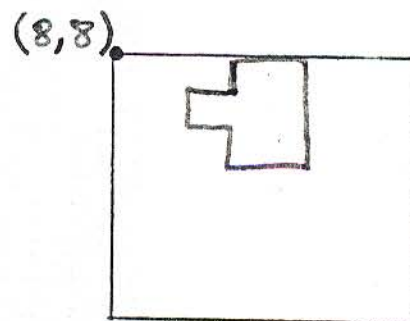
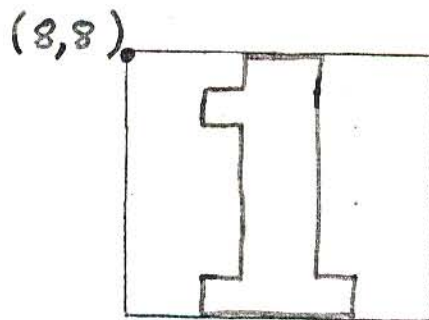
- The Y co-ordinate is only relevant for the fourth object in the group and the other 3 take on the same value
- The X co-ordinate of the first object is taken from the fourth object. The second third and fourth objects in the group have X-values of $X+16$, $X+32$ and $X+48$.

The same comments apply as regards LSS values, overlap, clipping and the co-ordinate ranges.

Displaying Partial Majors.

It is possible to display just the upper part of a Major (or Group Major). This is done by incrementing the LSS value by 1 for each row of the object to be rejected. At least two rows must remain! Another consideration is that the collision rule applies as if the whole object were present.

Example To display a whole and partial "1" at (8,8)
Keycode = 1.



} 4 lines
skipped

X	8
Y	8
LSS	4
colour	colour code

8
8
8
colour code

Memo: $LSS = (\text{keycode} \times 8) - Y/2$

Displaying user-described objects (MINORS)

Minors are more flexible and powerful than Majors.

The user defines the object pattern in 8 bytes of memory per Minor. There are 4 Minors available. Minors may collide with other Minors or with Majors.

Display of the minor takes priority over that of the Major, so that Minors appear on top of Majors. Also, the priority order amongst Minors themselves is 1 (highest) to 4 (lowest).

- Additional Features of Minors are
- Double Size display capability
 - Smoothing (shifts alternate lines of the pattern $\frac{1}{2}$ position)

Bit Pattern Lay-out for the definition of a Minor

	7	6	5	4	3	2	1	0
80								

The bit pattern has the same layout as the object pattern. A bit is set to 1 to denote that the corresponding pixel is to be displayed.

The address 80 refers to the first minor. For the addresses of the other minors, refer to the memory map in Appendix 2.

A Minor is further specified by a separate RAM-block:

	7	6	5	4	3	2	1	0
00	Y	Y	Y	Y	Y	Y	Y	Y
01	X	X	X	X	X	X	X	X
02			B	G	R	DBL	SM	X9
03								

Y = Y co-ordinate (may be odd or even).

X = X co-ordinate X9 gives a resolution of $\frac{1}{2}$ pixel.

SM = smoothing bit, (1=ACTIVE)

DBL = Double Size, enlarged about the top left hand corner (1=ACTIVE)

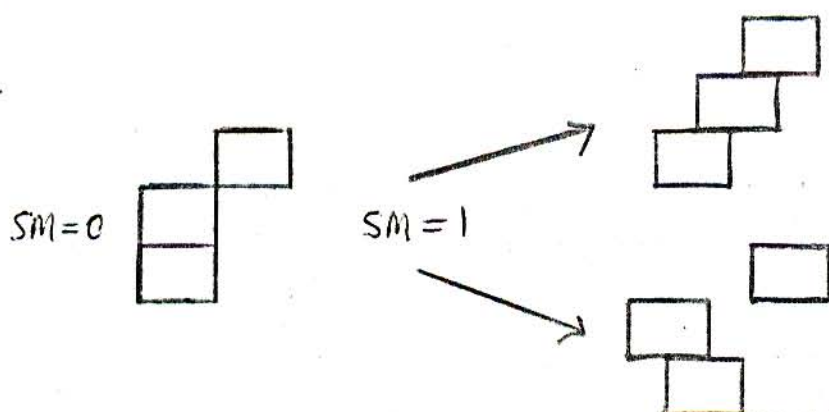
B = Blue colour component (1=ACTIVE)

G = Green colour component (1=ACTIVE)

R = Red colour component (1=ACTIVE)

Smoothing

Smoothing has the effect of displacing alternate rows of pixels half a unit to the left. According to the Y-coordinate alignment, the effect may be to smooth or disperse



Smoothing is useful for drawing edges to objects at oblique angles.

Minors have the same co-ordinate range as Majors.

There are no restrictions on colliding Minors—see the Section on "Movement of Objects"

Picture Layout and Co-ordinates

40

Figure 3.2 illustrates the relative positioning of the grid with respect to the TV screen edge. The co-ordinates as used for addressing Majors and Minors are also indicated.

The figure is based on experience with a PAL system.

Description

The background covers the whole screen, with the exception of the very top and bottom. The effective coordinates of the limits of the background are: in X -3 to 170 and in Y 0 to 242. Due to "overscan" the system may in fact be attempting to address a considerably broader area.

The grid extends to nearly the background limits. A narrow grid may appear slightly left of centre - this is because a wide grid requires more space to the right.

The coordinates of the limits of the grid are: in X 8 to 153/167 (narrow/wide grid) and in Y 24 to 218.

The figure also shows

- exactly what is meant by a vertical and horizontal grid segment
- the appearance of an 8x8 object centred in a grid cell
- the wide grid units.
- the meaning of a "dot"

The significant addresses are marked and explained on a following sheet.

Y ↑
0 -

24 -

48 -

2 grid segment dimensions

3 ⇄

← 14 →
width

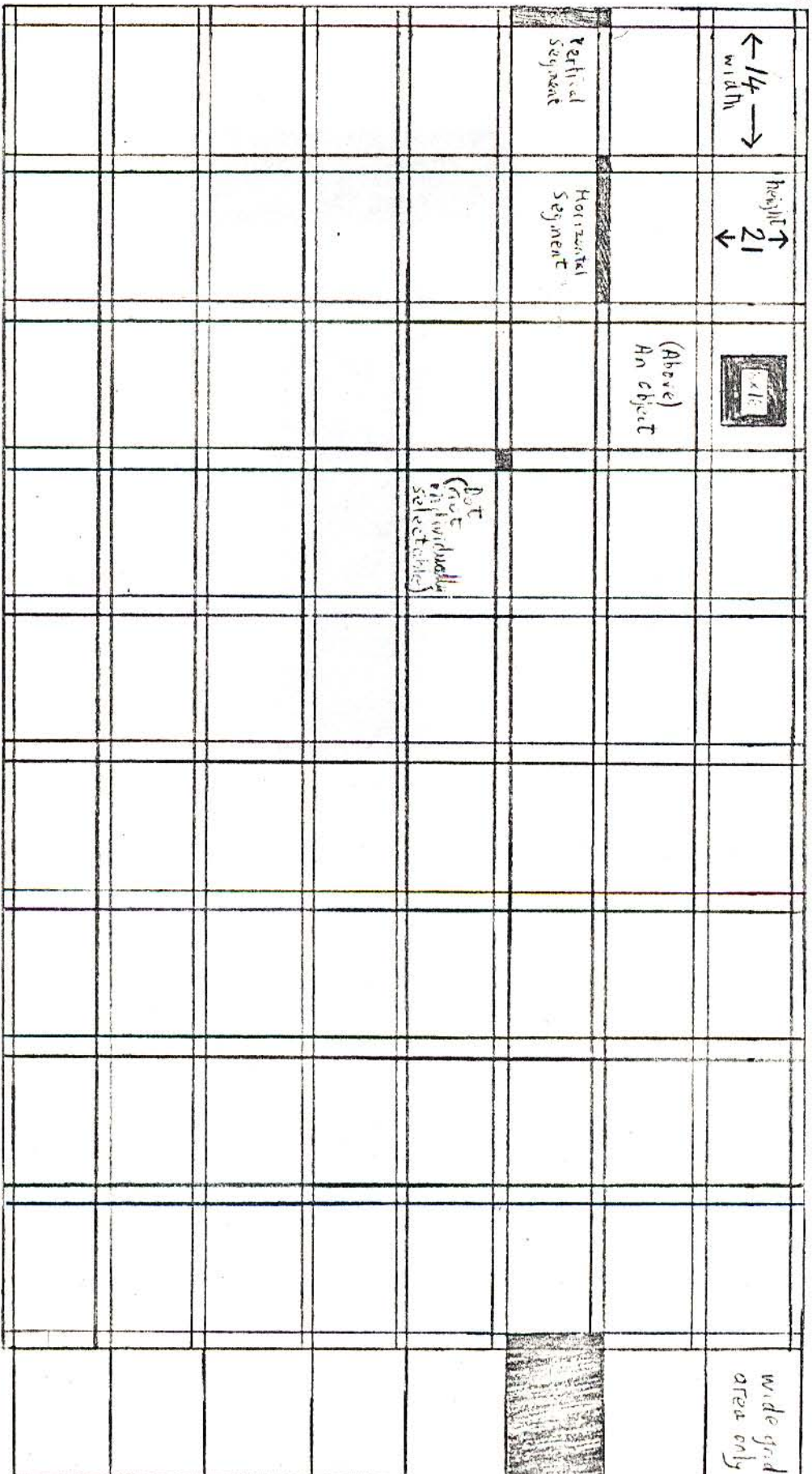
height ↑
21 ↓



(Above)
An object

Dot
(not individually
selectable)

wide grid
area only



242 -

227 -

218 -

3-10 8 9
X →

(C)

(C)

152 153 163 167 170

Significant Addresses (PAL)

$X = -3$	approx lower limit of background
$X = -1$	lowest physical X-value for an object, programmed by setting $X = 227$
$X = 0$	lowest X-value when using normal sequential X co-ordinates
$X = 8$ and 9	Addresses of the first vertical segment of the grid
$X = 152$ and 153	Addresses of the last vertical segment of the grid
$X = 163$	Highest address for an 8x8 object that is to be displayed whole.
$X = 167$	Address of the extreme right pixel of the wide grid segments
$X = 170$	Highest address whereby the extreme left of an 8x8 object is still just visible. Also the upper limit of background
$Y = 0$	Lower limit of background Lowest address of an object
$Y = 24, 25$ and 26	Addresses of the first horizontal segment of the grid
$Y = 216, 217$ and 218	Addresses of the last horizontal segment of the grid
$Y = 227$	Highest address for an 8x8 object that is to be displayed whole
$Y = 242$	Highest address whereby the top of an 8x8 object is still just visible. Also the upper limit of background.

Note for NTSC

Objects must remain within the limits of the grid, ie (top left of object)

Range in X: 8 to 160 inclusive.

Range in Y: 24 to 203 inclusive.

FIG 33 TABLE OF ACCEPTABILITY OF COLOUR COMBINATIONS OF AN OBJECT ON A BACKGROUND

OBJECT BACK- GROUND	BLUE	GREEN	RED	YELLOW	TURQUOISE	PURPLE	BLACK	WHITE
BLUE	✓				✓	$\frac{1}{2}$		$\frac{1}{2}$
GREEN		✓		$\frac{1}{2}$				
RED							$\frac{1}{2}$	$\frac{1}{2}$
YELLOW		$\frac{1}{2}$		✓				
TURQUOISE					✓			
PURPLE						✓		✓
BLACK	✓	✓	✓	✓	✓	✓	✓	✓
WHITE								✓

✓ = Good

$\frac{1}{2}$ = Half Acceptable

This section pursues the methods of the previous section to achieve special effects. It also covers collision detection.

Background.

The background can be split into horizontal bands by changing the background settings at the right part of the frame scan cycle. In order to make the setting change at the right time, it is necessary to appreciate the construction of the TV frame as set out in Section 1. The programmer makes use of an interrupt at the start of vertical blanking. This is the frame scan reference point. The interrupt is vectored to 402H in the games program.

The games program can then set the event counter to count N scan lines. The games program may wish to make various other changes to the display during vertical blanking. It should also return to the executive label INTSVC so that the executive can deal with standard activities such as sound table output, display packet output etc.

The important aspect now is that after N scan lines have elapsed, the event counter interrupt transfers control to location 404H in the games program. This is the time to make a change to the background colour. There are a few precautions, however. One should program a short delay to ensure that the change is made during horizontal blanking, otherwise, the transition will cause an unpleasant effect.

Strictly speaking one should disable the ^{display} during any change ⁴⁵ to the graphics RAM. In this case it is important not to allow objects to span the two background areas, as this would confuse the 8244/8245 and the object may be partly duplicated. In practice however it is possible to change the background colour settings without disabling the display.

The number of lines to count from the vertical blanking interrupt is dependent on the TV system employed. For PAL, one must allow 25 lines to get to the first line of the new frame and then consider the fraction of the 290-odd displayed lines that need to be skipped.

The Grid

Similar comments apply to the Grid. The Grid parameters can also be changed without disabling the display (unofficially).

Objects

Objects can be changed (i.e. in place, shape, colour etc) by writing the new description into the Graphics RAM. In this case it is essential to obey the rule about disabling the display first. Disabling the display deletes all objects, but not the background or grid, for its duration. In order not to cause disturbing effects on the screen and also because of the 8244/8245, it is important that no objects should be on the ~~the~~ scan line(s) that is(are) disabled.

It may be necessary to build into the game a reserved area where no objects ever go. 46

The effect of a moving man can be made by alternating Major objects 32 and 33 (or 34 and 35). In general several frames of any one situation will be displayed before making the change to produce reasonable speeds.

It is possible to re-use the object descriptions in the Graphics RAM to produce more objects by the timing technique. This is especially useful where more than four Minors are needed.

Movement in Y of a Major object implies a change in LSS-value. The executive routine LSSCAL can be used, but in many cases the change in Y/2 can be added to the relevant byte, without a change in the M.S.B.

For instance, the 5 figures of men in the internal character set can be moved to any location on screen without affecting the m.s.b of the LSS. Numbers can be put at any location below 90 hex, and the alphabet in a narrow band between C0 and D0 hex, all without changing the m.s.b.

Movement can be implemented by adding the increment to the Y-value, shifting it right once, and subtracting it from the current LSS. The carry can be ignored.

An object can be removed from the display by setting the Y-coordinate to 0F8H

Collisions

47

PHILIPS

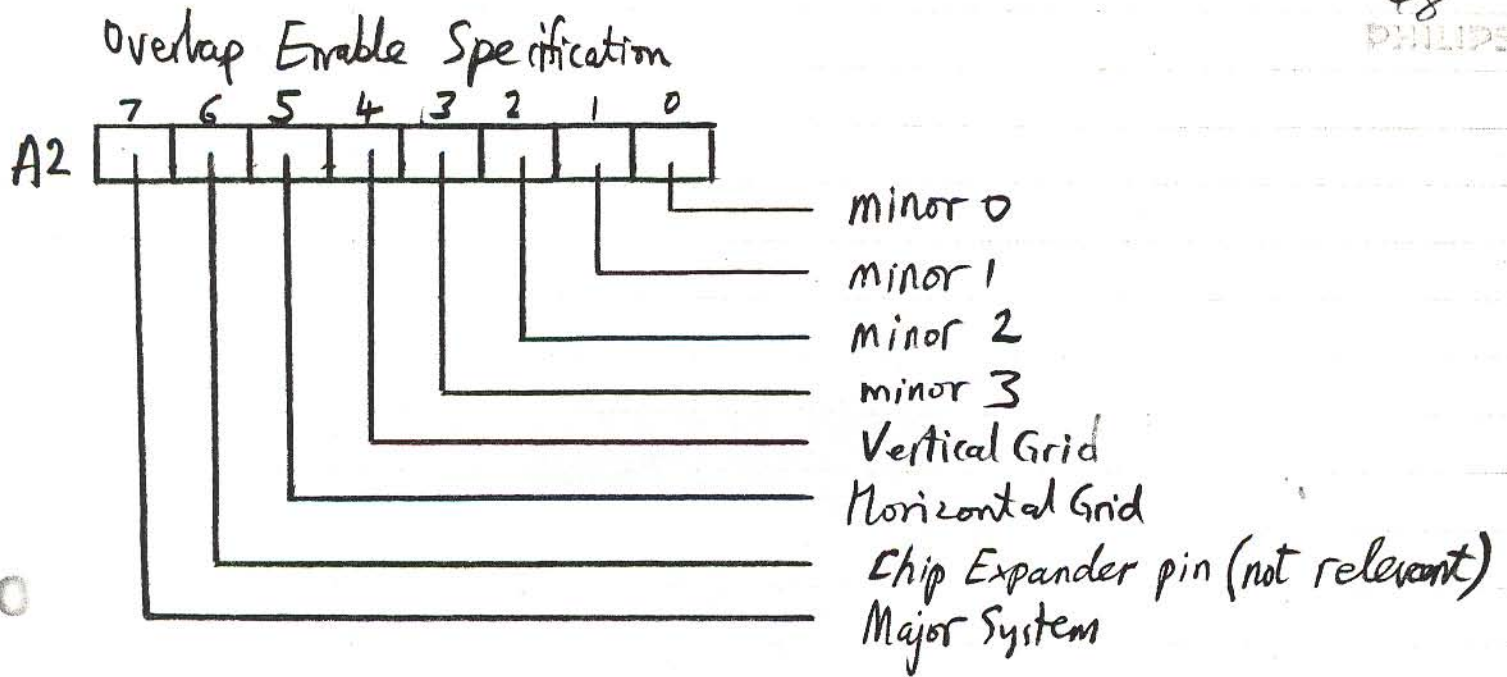
The 8244/8244 provides collision indicators. Graphics RAM location A2 is used for this function.

It was originally intended that Majors could overlap each other. This function could not in fact be implemented, and it is now the programmer's responsibility to ensure that this situation never occurs as it causes unpredictable effects on the screen.

The legal overlap combinations are:

	Major	Hor. Grid + Dot	Vert Grid	Minor 3	Minor 2	Minor 1	Minor 0
Major	NO	YES	YES	YES	YES	YES	YES
Hor. Grid + Dot		—	—	YES	YES	YES	YES
Vert. Grid			—	YES	YES	YES	YES
Minor 3				—	YES	YES	YES
2					—	YES	YES
1						—	YES
0							—

The user specifies which items are to be checked for overlap and is told by the system which items are involved in an overlap. It is then up to the user to sort out exactly what overlaps what in the case of several items being involved.



The relevant bits are set to 1 to indicate that overlap for this (type of) object is to be tested for. The setting should take place before a scan, i.e. during vertical blanking but before scanning the new frame.

The status is set as it is detected. A one indicates that the relevant object is involved in overlap. where only two objects are involved the indication is all that is needed. In other cases the program may have to apply some knowledge of the game situation to determine what has happened. The status remains valid until shortly after vertical blanking. (?)

The Strobe Function

49

PHILIPS

Graphics RAM byte A0 contains a bit for requesting the x and y coordinates of the beam position. It is bit 1 ("Forced Position Stroke"). The y and x coordinates are then put in locations A4 and A5 respectively and bit 1 of byte A1 is brought low when this has taken place.

5 THE GAME PLAYERS' PERIPHERALS

50
PHILIPS

The game players' peripherals are:

- 2 joysticks, each with an action button
- An alphanumeric keyboard.

These devices are for input only. It is possible to test for input at any time, but these devices are noninterrupting.

The keyboard

In practice one reads the keyboard by calling the executive routine KEYBRD. Use of this routine is especially recommended as it rejects transient phenomena which are caused by (de) bouncing.

The low-level operation of keyboard input is as follows:

Port 2 is used to address the keyboard. The keyboard must be enabled (via port 1). The 6 rows of the keyboard must be scanned by selecting them and then reading the data. A row is selected by outputting to bits (2, 1 and 0) of P2.

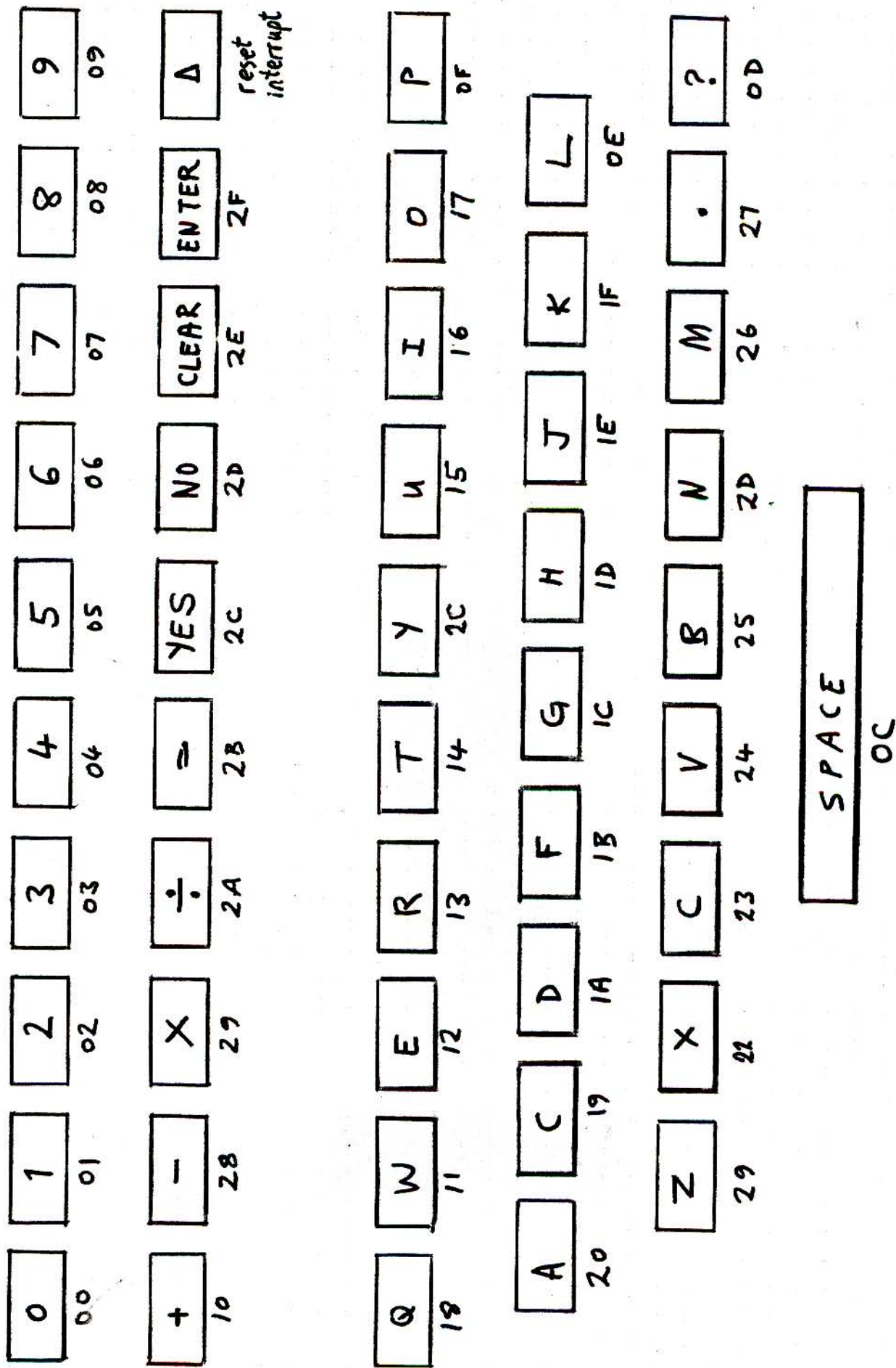
The port is then read
see if

and bit 4 is examined to see if there is valid data in this row. "Valid" here means that the key is depressed at the instant of reading the port. Bits (7, 6, and 5) contain the column code of the depressed key.

The routine KEYBRD scans all 6 rows. It combines the row and column numbers of any depressed key into a standard keycode. It also indicates no change/no key depressed.

For the exact description^{of KEYBRD}, refer to Section 8.

Fig 5.1) Keyboard with codes (hex) as used by the executive.



Note. The "Row" and "Column" designations in Port 2 do not correspond to the physical layout of the keyboard.

The joysticks

In practice one calls the routines RESLV and CALHD to obtain joystick and action button input. Refer to Section 8 for details.

The low-level operation is as follows:

The hand controls are enabled via port 1.

The required joystick is selected by outputting to port 2, bit

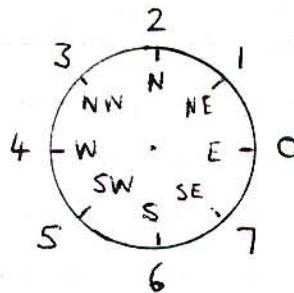
Joystick and action button data are read from the BUS.

All bits here are active low, as follows

4	action button
3	left
2	down
1	right
0	up

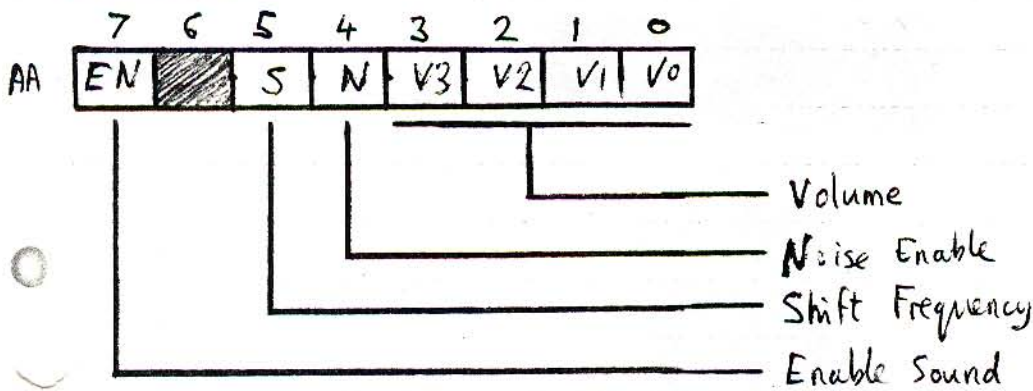
There are 8 positions of the joystick; the above plus upper-left upper-right lower-left and lower-right.

RESLV and CALHD calculate a "heading code" for a joystick according to the following convention



The 8244/8245 chip drives a sound generator using data specified in the Graphics RAM. Four bytes are reserved for this, a control byte and three data bytes.

Control word



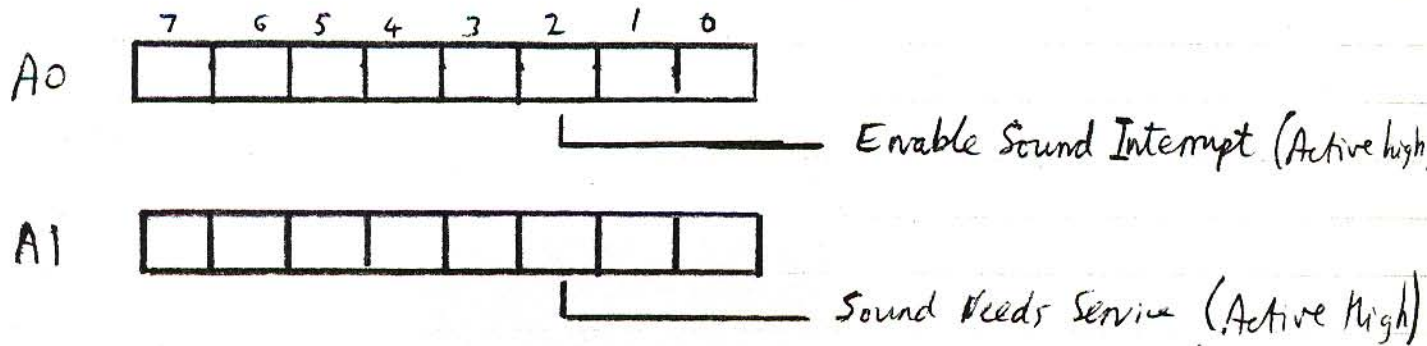
The volume bits define ^{one of 16} audio output levels 0000 = lowest, 1111 = highest

Bit 4 (Noise Enable) controls noise generation 1 = noise on 0 = noise off

Bit 5 (Shift Frequency) determines the frequency of the shift clock
 1 = 3933 Hz 0 = 983 Hz. For short, the frequencies are called 4KHz ^{and 1KHz}
 It is the three data bytes that are shifted. More about this below.

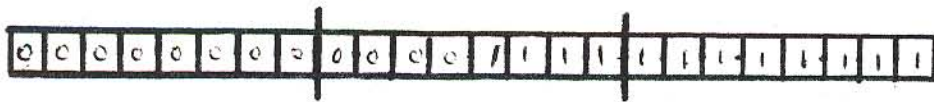
Bit 7 (Enable Sound) 0 = No Sound 1 = Sound.

The data bytes are at addresses A7 to A9. These define the bit pattern that is shifted circularly to obtain an audio frequency. The shift frequency is the rate at which the pattern is shifted one bit position. The 8244/8245 shifts indefinitely or until the sound is disabled. It is possible to generate an interrupt when all the 24 bits have been shifted once, so that a new bit pattern can be loaded. The control bytes for this are A0 and A1, the relevant bits being as follows



- A sound interrupt is identified as such by the Sound Needs Service bit in ^{external} byte A1.
- There are other interrupting sources such as vertical blanking. There is of course no reason why a sound interrupt should not occur during vertical blanking. It is vital that all interrupting devices are checked for and dealt with after ONE read of location A1 as this clears the interrupt request. (In many programs it would be unacceptable to miss a vertical blanking interrupt due to servicing of sound).

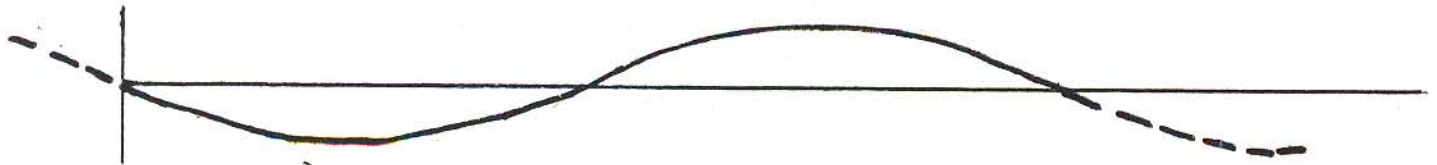
Tones which are subharmonics of the shifting frequency are easily produced. For example, a tone of 42 Hz can be produced with a 1 KHz shifting frequency and the following bit pattern:



Repeated shifting generates the following waveform



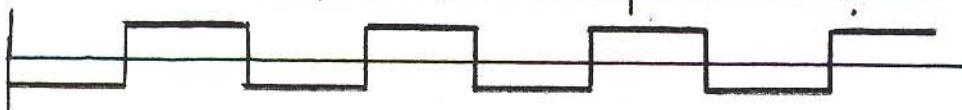
Passing the above through a low pass filter emphasizes the basic sine component:



↑
24 milliseconds

$$\text{frequency} = \frac{1}{0.024} \approx 42 \text{ Hz}$$

A tone of 667 Hz can be similarly generated with a 4 KHz shifting frequency and the following bit pattern

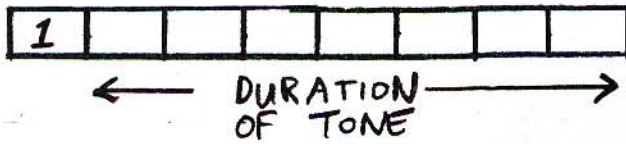


The executive contains 4-byte blocks for producing subharmonic tones at full volume, without noise. The tones are 42, 83, 125, 167, 250, 333, 500, 667, 1000, 2000 Hz.

The executive also defines "Sound Tables" whereby a list of tones and durations is specified. The duration is in frames (and as such is dependent of the TV system in use). The executive does not use sound interrupts.

Structure of Executive Sound Tables

Format 1, One tone for a given duration

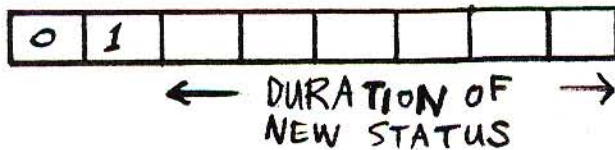


low order address of tone definition

Notes

The tone is taken from the standard executive tone definitions.

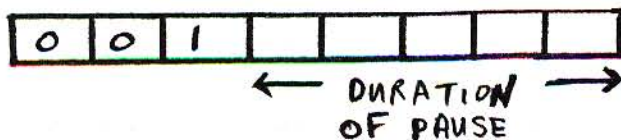
Format 2, Sound status change



← NEW SOUND STATUS →

Varies the ^{control} status word without altering the tone. Also used to produce noise.

Format 3, Pause



Format 4, Chaining



← NEW SOUND TABLE ENTRY ADDR →

Used to stop a sound temporarily, or, before the end of the table, permanently.

Used for looping and also jumping to a section of an existing sound table.

Format 5, End of Table

0	0	0	0				
---	---	---	---	--	--	--	--

Defines the end of the table.
Does not in itself cause the
last sound to be stopped.

The standard sound tables are on page 3 so that the MOVPS instruction can be used to access the entries via a pointer.

The executive contains sound tables for
a crash (label CRASH)
a gunshot (label GUNSHOT)
and others.

~~The user can define his own tables and place them in RAM and call the executive to process them.~~
The routine is SNDSET (see the section on executive routines for details).

The executive jumps to 40AH when an entry in the sound table has expired. The user can then simply call INTSND for standard executive sound tables. When the entire sound table has been worked through, the sound flag (bit 6 of MISC in Data RAM) will be cleared.

For user defined tables the user must step on to the next entry in the table in a routine at 40AH. The call is then to EXSETY, alias EXTSND. The reason that the executive cannot do this is that ROM tables other than those on page 3 cannot be accessed by the executive.

For details of the EXSETY call, refer to the section on executive routines.

Section 7. TYPICAL PROGRAM STRUCTURE

60
PHILIPS

A games program typically consists of the following components

Linkage to the Executive ← User's Symbols
Startup code

A Vertical Blanking Interrupt Routine

A Sound Table Service Routine

General Processing, Preparation of new screen layout

Input from Keyboard and Joysticks

An idle Routine

More advanced games may need

A Timer/Counter Service Routine

A Sound Interrupt Routine

These topics are dealt with one by one, and summarized in the case of executive and user defined service routines.

Finally, a view of game processing with respect to the TV frame cycle is given.

Linkage to the Executive

61
PHILIPS

The first part of a games program should be a list of equates defining executive entry points. The symbols used should be those used in the standard executive listing. These symbols should preferably be included by a macro or an assembly pre-pass so that the accuracy of them can be guaranteed.

Examples

INTSVC	EQU 009H	EXT. INTERRUPT SERVICE
VRTETY	EQU 01AH	VERTICAL BLANKING SERVICE
INTSND	EQU 044H	EXEC SOUND TABLE SERVICE
etc.		

Executive routines are then called or jumped to using these symbols. Other entry points in the executive should be referenced symbolically too e.g. the names of the sound tables. Various other executive-defined symbols can also be included for use in the games program, e.g. symbols for masks for I/O operations names of reserved bytes in the various RAMS

= 110 ;

= 111 ; COLOUR DEFINITIONS (MINORS)

= 112 ;

0000 = 113 BLAMIN EQU 00H ; BLACK
0020 = 114 BLUMIN EQU 20H ; BLUE
0010 = 115 GREMIN EQU 10H ; GREEN
0028 = 116 PURMIN EQU 28H ; PURPLE
0023 = 117 REDMIN EQU 08H ; RED
0030 = 118 TURMIN EQU 30H ; TURQUOISE
0036 = 119 WHIMIN EQU 36H ; WHITE
0018 = 120 VELMIN EQU 18H ; YELLOW

= 121 ;

= 122 ; COLOUR DEFINITION (MAJORS)

= 123 ;

0000 = 124 BLAMAJ EQU 00H ; BLACK
0008 = 125 BLUMAJ EQU 08H ; BLUE
0004 = 126 GREMAJ EQU 04H ; GREEN
0007 = 127 PURMAJ EQU 07H ; PURPLE
0002 = 128 REDMAJ EQU 02H ; RED
0000 = 129 TURMAJ EQU 0CH ; TURQUOISE
000E = 130 WHIMAJ EQU 0EH ; WHITE
0006 = 131 VELMAJ EQU 06H ; YELLOW

= 132 ;

= 133 ; POSITION DEFINITIONS

= 134 ;

0018 = 135 XPOS1 EQU 18H ; X VALUE OF TOP LEFT CORNER LINE 1
0020 = 136 YPOS1 EQU 30H ; Y VALUE OF TOP LEFT CORNER LINE 1
0018 = 137 XPOS2 EQU 18H ; X VALUE OF TOP LEFT CORNER LINE 2
0050 = 138 YPOS2 EQU 6CH ; Y VALUE OF TOP LEFT CORNER LINE 2
0018 = 139 XPOS3 EQU 30H ; X VALUE OF TOP LEFT CORNER LINE 3
0098 = 140 YPOS3 EQU 0A3H ; Y VALUE OF TOP LEFT CORNER LINE 3
000A = 141 XMIN EQU 0AH ; MINIMUM X VALUE FOR MAJOR/MINOR
0032 = 142 YMIN EQU 32H ; MINIMUM Y VALUE FOR MAJOR/MINOR
0087 = 143 XMAX EQU 87H ; MAXIMUM X VALUE FOR MAJOR/MINOR
00A7 = 144 YMAX EQU 0A3H ; MAXIMUM Y VALUE FOR MAJOR/MINOR

Users Symbols

63

PHILIPS

It is a good rule ALWAYS to use symbols as operands to instructions. This eases program maintenance in the future because it means that any one quantity is only defined once, and is easy to find (from a symbol table list or cross-reference list). This is especially important where future changes or changes during development are foreseeable e.g. RAM layout, game parameters.

Certain symbols are needed in almost every game and can best be standardized. The following are included on the following page as a start

- colours of minors
- colours of majors
- limiting screen position definitions

The user should define the layout of internal data RAM and external data RAM using symbols for the byte addresses. Further equates will generally be useful for constants e.g. to be used as immediate operands.

Startup Code.

64
PHILIPS

When the game-player plugs in a cassette after powering up, the screen will be probably be arbitrary. The game-player presses RESET to start up the game. The effect of this for the programmer is a jump via address 0 to address 400H.

The programmer will have to jump to a new address as address 402H is reserved for a different vector (external interrupts).

The startup code may wish to call the executive routines CLEAR (initialize RAMs) or START (initialize RAMs and output SELECT GAME and read keyboard).

Vertical Blanking Interrupt Routine.

65

This routine will take advantage of the fact that nothing is being displayed at the time and disable video and move data into the Graphics RAM.

The standard executive routines for vertical blanking are INTSVC, VRTETY and INTSND. These perform the following actions

- maintain a modulo 60 counter
- move data packets from External to Graphics RAM
- maintain sound as necessary.

The user may wish to add extra functions such as starting the (TV-line) counter for an interrupt during the next frame scan.

In some applications the user may process the entire vertical blanking servicing in the games ROM program.

The alternatives are illustrated in the summaries at the end of this section.

For normal purposes the executive routine will suffice, working with either executive-defined sound tables or user-defined ones. The executive examines the progress of sound effects at vertical blanking time. The result can be

- no sound in progress
- sound to be continued without alteration, as the counter has not expired
- next entry in the sound table to be loaded
- sound servicing to be stopped

The routine for doing this is INTSND for exec-defined sound tables and EXSETY for user-defined sound tables. In the latter case the caller must load the sound table entries for the executive (to which they are inaccessible directly) prior to calling — see the description of EXSETY for details.

If the user defines his own format of sound tables, but works on the same principle as the executive (sound control for a given duration measured in vertical blanking counts) then INTSND/EXTSND can be replaced by a user routine. If the user works with sound interrupts then the routine INTSVC must be replaced as well. ~~FE~~

General Processing, Preparation of New Screen Layout

67

General processing is that which does not involve any I/O or access to the Graphics RAM. Typically this involves working out new co-ordinates for moving objects, working out where new objects must appear, which existing objects must disappear or change in shape etc. This kind of processing should be done at main program level (as opposed to interrupt level). Moreover, the processing should be completed by the next vertical blanking interrupt. If necessary, arrange for different aspects to be dealt with on different series of frame scans so that each one aspect has enough time to complete. e.g. on every odd frame scan deal with player 1 and on every even scan with player 2.

It is possible to process independently of vertical blanking interrupts. In this case bear the following in mind

- don't use flag F1.
- turn interrupts off where it is important not to be interrupted with half-complete operations (such as half complete data packets)

This method of working is not recommended.

The way to regain control after vertical blanking servicing is to work in a loop, calling `NXTINT` ~~at the~~ when processing for the current frame is complete. This structure is shown in the summaries in this section. The time between gaining control from `NXTINT` to re-calling it with general processing complete should not exceed about 15 milliseconds.

Input From Keyboard and Joysticks

89

PHILIPS

A good time to read the keyboard and joysticks is at main program level just after vertical blanking servicing, i.e. after coming out of the NXTINT routine.

At this time the display may be active so it is useful to make constructive use of the time in this way. Also, the results can be used in the processing which follows, (calculations etc)

Make use of routines KEYBRD, RESOLV, CALHD for keyboard and joystick I/O.

Idle Routine

69
PHILIPS

The standard idle routine is `NXTINT`. This routine should be called, with the video in the enabled state, when processing for the current frame cycle is complete. Control is returned after vertical blanking servicing.

Summary of Program Structure

Using Executive Routines for Graphics RAM Data Transfer, Sound Servicing etc.

70
PHILIPS

— equates —

(source caller)

400	JMP	START	(RESET)	RE-INIT + "GAME SELECT"
402	JMP	INTSVC	(EXT. INTERRUPT)	GO SERVICE EXT. INT.
404	—		(Timer not used)	
406	JMP	VRTETY	(INTSVC)	GO SERVICE VERT. BLANK.
408	JMP	GSTART	(START)	GO LAUNCH GAME
40A	JMP	INTSND	(VRTETY)	GO SERVICE SOUND

GSTART EQU \$

GAMELP CALL NXTINT

CALL DISABL

{ modify Graphics RAM further while vertical blanking lasts)

CALL ENABLE

(Read Joysticks and/or Keyboard)

(Do calculations)

CALL OUTPUT

CALL SNDSET

JMP GAMELP

idle loop

disable display

enable display

set up new Graphics RAM data

set up new sound

don't spend more than 15 msec

getting here!

Summary of Program Structure

using user-routines for Graphics RAM Data Transfer, Sound Servicing etc.

— equates —

```
400    JMP    GRESET
402    JMP    EXTINT
404    JMP    TINTSV
```

GRESET EQU \$ GAME RESET ROUTINE
(Initialize Game)

GAMELP CALL NXTINT
Read Keyboard and/or Joysticks.
Do Calculations. Prepare Graphics Data.
JMP GAMELP

EXTINT EQU \$
Save Registers. CALL DISABL
Determine the cause(s) of the interrupt. This can
be sound or vertical blanking. Call the relevant
routine(s), SOUTINT/VBLINT.
Restore registers. CALL ENABLE and return.

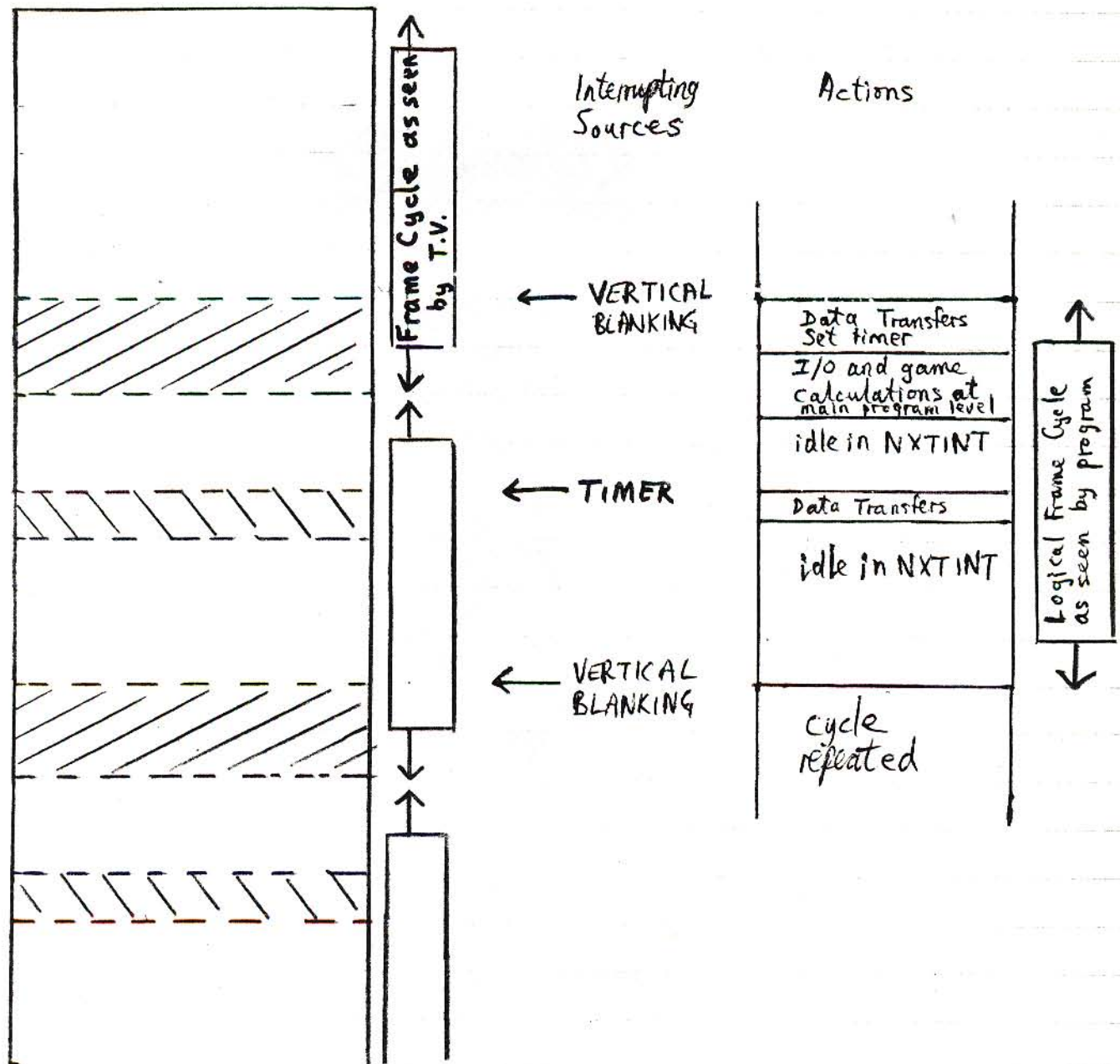
VBLINT EQU \$
Read control and status bytes
Transfer Data to Graphics RAM
Set Timer

SOUTINT EQU \$
Re-load sound registers

TINTSV EQU \$
Save Registers, CALL DISABL, perform similar actions to EXTINT/VBLINT
as needed, Set timer again, restore registers, call ENABLE and return.

Game Program Structure as seen with respect to The T.V. Frame Cycle

72
PHILIPS



Shading indicates
DISABLED VIDEO

/// vertical blanking period,

/// user disabled video period,

8. Executive Routines

74

PHILIPS

Contents

Register, RAM, and Flag usage in The Executive
Reset and Interrupt Processing: using The Executive
Basis for better interrupt processing

BLKT1 and BLKT2

CALHD

CLEAR

CLRMAJ

CLRMJ

DSABL

DVDE

ENABLE

EXTRAM

EXSETY (=EXTSND)

GEN

GEN1

HANDCTL

INTSND

INTSVC

KEYBRD

KEYCHK

LDMAJ

LSSCAL

MULT

NXTINT

OPMJ

OTFGST

OUTPUT

OUTP

Block Transfer Select Masks

Calculate Joystick Heading

Initialization of RAMs.

Clear all single Majors

Fill RAM with a constant

Disable Video and Grid

Divide

Enable Video and Grid

Select External Data RAM

Sound Table Handling (user-defined tables)

} Calculate LSS and move bytes 3 and 4
of a Major object to a Data Packet.

Hand Control Enable Mask

Sound Table Handling (exec-defined tables)

External Interrupt Service Routine

Scan Keyboard for key entered

Wait for keyed input and make keep sound

Calculate LSS of Major Object and
load into Graphics RAM

Calculate LSS value of Major Object

Multiply

Wait for next Vertical Blanking Interrupt

Calculate LSS of Major Object and load
entire Major into Graphics RAM.

Output Flag Set and Terminate
Data Packet

Move Data Packets from Ext. RAM to Graphics RAM

Output Time

OUTPTI
RDBT
RESLV
RNDM
RNDM2
SCORE
SCROUT

SDBT
SL8244
SNDSET
START

TDBT
TIMER
TIMINT
VRTETY

Move Data Packets from Ext. RAM to Graphics RAM⁷⁵
Reset (= clear) a bit in External RAM or Graphics RAM
Get status of Joystick
Random Number Generator
Random Number Generator using Timer
Output score digits
Calculate LSS and Output Major Object to
Data Packet
Set a bit in External RAM or Graphics RAM
Select Graphics RAM
Initialize Sound Table Processing
Start of Game Selection and initialization
Sequence
Test a bit in External RAM or Graphics RAM
Maintain the time on screen
Initialize Timer
Vertical Blanking Interrupt Service Routine

Register, RAM and Flag usage in The Executive.

Register Usage.

76 PHILIPS

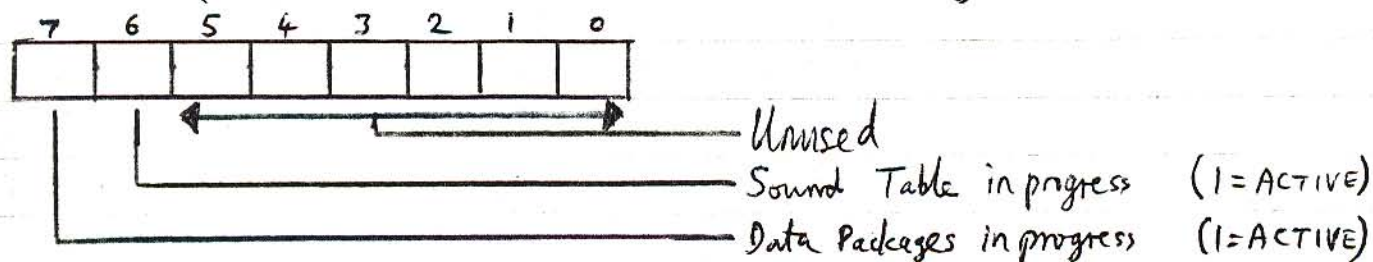
The executive uses register bank 0 (RBO) for interrupt routines and routines synchronized with vertical blanking interrupts.
Some registers have specific functions

R0	general
R1	general
R2	general
R3	sound table pointer
R4	count for sound duration
R5	ACC save location
R6	P1 save location
R7	last keyed in character

Reserved Words in RAM

<u>Internal RAM</u>	63	MISC
	62	MODU60
	61	OVSTAT

MISC (miscellaneous) contains the following flags



MODU60 is a ^{binary} modulo 60 frame counter. In the NTSC implementation this is a seconds counter. Includes stop bit and inc bit.
Bit 6 = inc bit = parameter to routine TIMER. Bit 7 = stop bit for routine TIMER

OVSTAT

The executive saves the overlap status word (read from the Graphics RAM) in OVSTAT in the vertical blanking interrupt routine.

External RAM

127	OUTADS
2	SECNDS
1	MINTES
0	MISC2

77 PHILIPS

OUTADS (output address)

This location is used by the executive routine OUTPUT. It is read and taken as the pointer to data packets to be transferred to the Graphics RAM.

SECNDS	(BCD seconds counter)
MINTES	(BCD minutes counter)

The above two locations are entry requirements for the executive timer routine. These are then maintained by the timer routine, and used in preparing a data packet to the Graphics RAM for timer display.

MISC2 (miscellaneous flags byte 2)
unused.

Flag FI

This is an interrupt flag, set during vertical blanking interrupt processing and cleared by a call to `NXTINT`, a routine which can be called by a main program to wait for the next vertical blanking interrupt. After servicing the interrupt the executive will return control to the caller of `NXTINT`.

78 PHILIPS

In ^{USER} interrupt routines which are activated on the vertical blanking interrupt, FI should be set before returning.

The games programmer is free to use FI as he wants provided the main program does not get interrupted by vertical blanking interrupts. There is no danger of this if the timing in the program is correct (as already discussed).

ACC, C

The accumulator is used in practically every routine and will not generally be restored on return. An exception is that interrupt routines other than vertical blanking returning to the interrupted code restore ALL registers etc. so that the interrupted code can continue unaware that it was interrupted. C (carry) should also be considered unreliable after calling a subroutine.

Reset and Interrupt Processing using the Executive 79 PHILIPS

The hardware makes a jump to location 0000 on a reset, and a call to 0003 or 0007 on an interrupt. These locations are in the executive and are vectored to the games ROM as follows

0000	JMP	400H	Reset
0003	JMP	402H	External Interrupt
0007	JMP	404H	Timer/Counter Interrupts

The games ROM layout for standard servicing is then

0400	JMP	(start of game)	
0402	JMP	INTSVC	executive external interrupt routine
0404	JMP	(timer/counter)	
0406	JMP	VRTETY	executive vertical blanking interrupt
0408	JMP	(start of game)	
040A		(sound)	

The JMP VRTETY processes vertical blanking interrupts. The routine INTSVC makes the jump to 0406 after having identified the interrupt as vertical blanking.

The routines INTSVC and VRTETY are described separately.

Owing to shortcomings in the above routines an alternative basis for interrupt servicing is given in the next subsection.

Basis for better interrupt processing

80 PHILIPS

0003 JMP 402H

HARDWARE EXTERNAL INT. CALL HERE

0402 JMP EXTINT

DEAL WITH EXTERNAL INT IN GAMES ROM

EXTINT SEL M80

MOV R5, A

SAVE ACC

IN A, PI

MOV R6, A

SAVE PI

CALL SL8244

MOV R0, #CTSTAT

READ STATUS AND CLEAR INTERRUPT REQ

MOVX A, @R0

MOV R0, #SAVCTS

(= USER RAM LOCATION)

MOV @R0, A

SAVE IN RAM

JB2 SOUT INT

SERVICE SOUND

SOUT RET MOV R0, #SAVCTS

MOV A, @R0

GET STATUS AGAIN AS PREVIOUSLY READ

JB3 VBLINT / VRTETY

SERVICE VERTICAL BLANKING SELF/EXEC

JMP RESTOR

GO RESTORE AND RETURN

VBLINT

=

CIR
CPL

FI
FI

JMP RESTOR

DO DATA PACKET TRANSFERS
UPDATE COUNTERS
SET FI FOR NXTINT ROUTINE

SOUT INT

=
=

JMP SOUT RET

SERVICE SOUND ACCORDING TO TABLE

BLKT1/BLKT2

Block Transfer Select Masks

81

2111123

Block transfer select

(No routine in executive; included to show use of masks).

X	X	X	X	X	X	X	X
---	---	---	---	---	---	---	---

Port 1

0	1	1	1	1	1	0	0
---	---	---	---	---	---	---	---

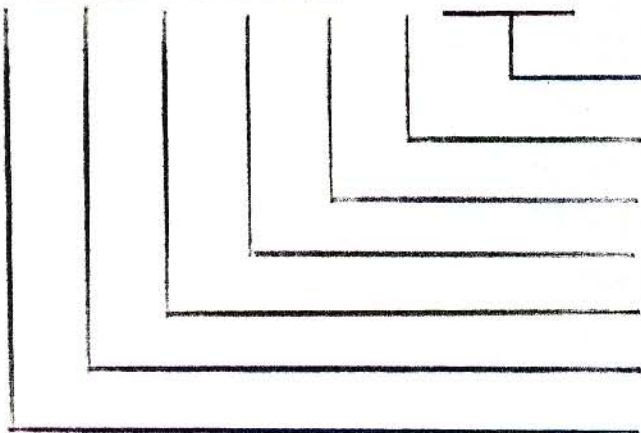
ORL P1, #BLKT1

X	1	1	1	1	1	X	X
---	---	---	---	---	---	---	---

1	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

ANL P1, #BLKT2

X	1	1	0	0	1	X	X
---	---	---	---	---	---	---	---

Games ROM unalteredKeyboard and hand controls disabledGraphics RAM enabledExternal RAM enabledHand controls disabledOne way transfer selectedForce video high unaltered

Routine CALHD

82

Function Calculate Joystick Heading

Address 3B1H

Presettings

Postsettings

Input Parameters R2 = Δx as obtained by RESLV
R3 = Δy as obtained by RESLV

Output Parameters R1 = direction.

Registers Altered R0, R1

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

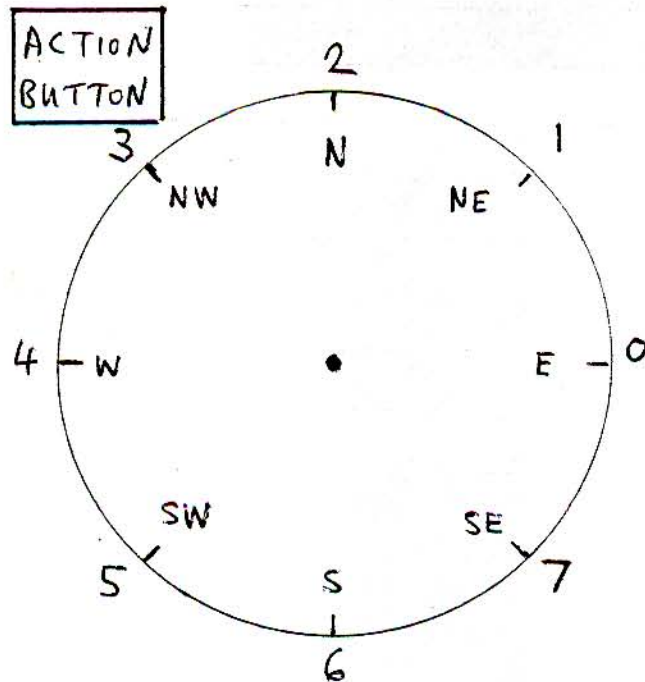
Called By

CALHD

Calculate Joystick Heading

83

This routine works with the output of the RESLV routine (R2 and R3) to produce a heading code as follows:



The routine assumes that some heading is present (joystick not centred).

Routine CLEAR

Function Initialization of Internal RAM, External RAM, Graphics RAM

Address 0F1H

Presettings

Postsettings 8244 selected, interrupts enabled.

Input Parameters

Output Parameters

Registers Altered R0: R0 R1: R0, R1

8048 Control Read

8048 Control Altered PI (see SL 8244)

Internal RAM Read

Internal RAM Altered 20 to 3F (cleared to 0)

8244 RAM Read

8244 RAM Altered 00 to 7F (set to 0F8H), 80 to FF (to 0)

External RAM Read

External RAM Altered 00-7F (cleared to 0)

ROM Data Read

Calls

Called By

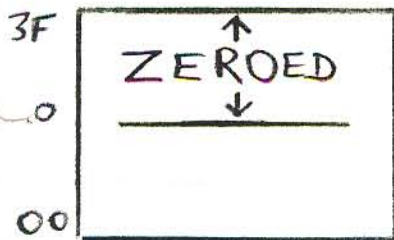
EXTRAM, SL8244, DISABL, ENABLE
START

CLEAR

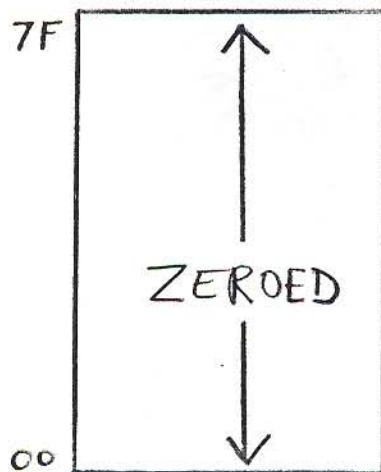
Clear RAMs

85 PHILIPS

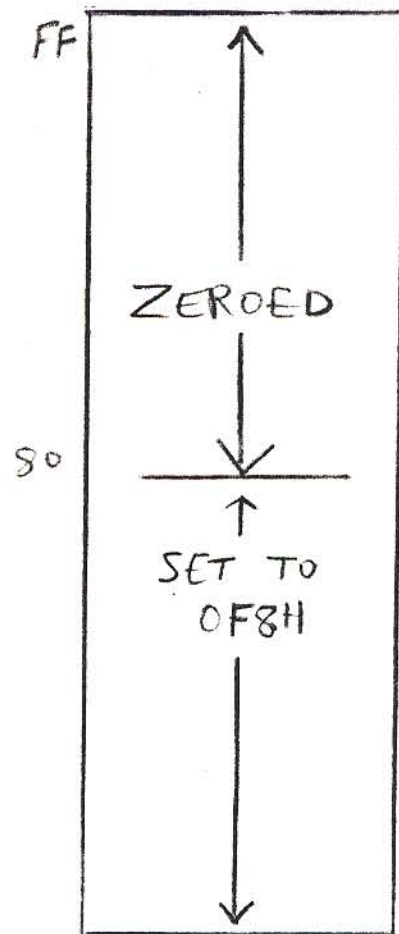
This routine can be called on game start up to clear all RAM. The lower half of Graphics RAM is set to 0F8H, as this implies a Y-co-ordinate that effectively removes the object.



INTERNAL RAM



EXTERNAL RAM



GRAPHICS RAM

Routine CLRMAJ

86

Function Clear all 12 Single Majors

Address 16BH

Presettings Call SL8244 and DISABL

Postsettings

Input Parameters

Output Parameters

Registers Altered RBI: R0, R2

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered All single majors

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By START

Routine CLRMJ (alias UNIFIL)

Function Copy ACC contents into consecutive bytes of RAM
(Graphics or Ext. Data).

Address 171H

Presettings Call EXTRAM or SL8244

Postsettings

Input Parameters ACC = Value to be propagated
R0 = RAM Area start address.
R2 = Number of bytes to be filled

Output Parameters

Registers Altered RBI: R0, R2

8048 Control Read
8048 Control Altered

Internal RAM Read
Internal RAM Altered

8244 RAM Read
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

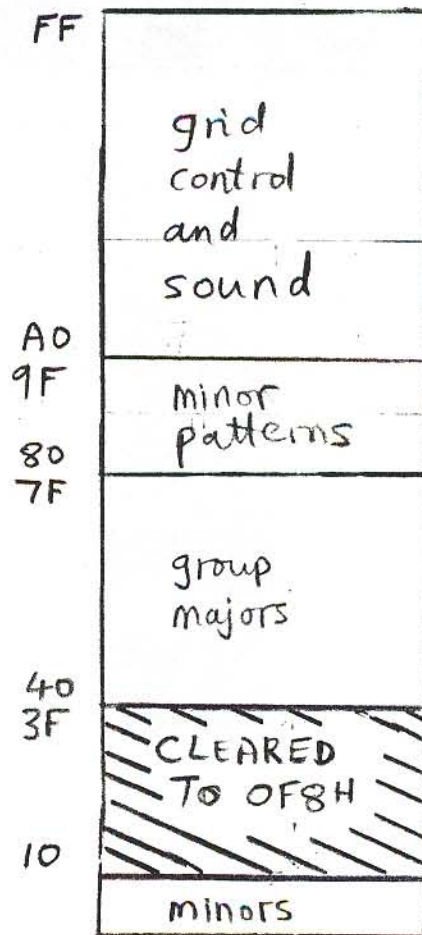
Calls
Called By CLRMAT

CLRMAJ, CLRMJ (=UNFIL) Clear all Single Majors in The Graphics RAM

88

The twelve single majors are cleared by putting 0F8H in each location involved. This acts as a safe off-screen y-coordinate.

The part of memory cleared is illustrated by a memory map.



The calling program must have selected the Graphics Memory and disabled the video display before calling CLRMAJ.

The entry point CLRMJ, alias UNFIL, can be used where the user defines the limits of RAM and the value to be filled in the RAM. This routine is then (sensibly) usable with External Data RAM as well.

Routine DISABL
 Function Disable Video and Grid
 Address IICH
 Presettings Select 8244 (SL8244)
 Postsettings Interrupts enabled

Input Parameters

Output Parameters

Registers Altered RBO: RO

8048 Control Read
 8048 Control Altered

Internal RAM Read
 Internal RAM Altered

8244 RAM Read
 8244 RAM Altered

External RAM Read
 External RAM Altered AO

ROM Data Read

Calls
 Called By Various

DISABL

Disable Video

90 PHILIPS

This routine switches off interrupts on entry so as to work with register bank zero. It disables horizontal interrupts, grid and display. This function is necessary when moving objects, although there is no need to disable the grid. As mentioned in previous Sections, one should not disable on a scan line which itself contains objects.

On return, the routine re-enables interrupts.

Note the presetting requirement that the Graphics RAM must have been selected

One might consider calling the routine at $DISABL+2$ so as to work without disabling interrupts and with the current register bank.

Routine DVDE

Function Divide

Address 3CFH

Presettings

Postsettings

Input Parameters R2 = dividend R3 = divisor

Output Parameters R4 = Quotient (= $R2/R3$) R5 = Remainder.

Registers Altered R4, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

DVDE

Divide

92

This routine performs unsigned division of one byte by another by repeated subtraction. Both the quotient and remainder are available as results. Overflow is not possible.

An attempt at division by zero causes an infinite loop!
One can prevent this by coding

```
MOV A, R3  
JZ ERR  
CALL DVDE
```

Bear in mind that dividing high numbers by low ones is relatively time consuming.

Routine ENABLE

Function Enable Video and Grid

Address 127H

Presettings Select 8244 (SL8244)

Postsettings Interrupts enabled

Input Parameters

Output Parameters

Registers Altered RBO: R0

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered A0

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By Various

ENABLE

Enable Video

94

This routine switches off interrupts on entry so as to work with register bank zero. It enables the grid and display, although not horizontal interrupts. On return, the routine re-enables interrupts.

Note the presetting requirement that the Graphics RAM must have been selected.

One might consider calling the routine at $ENABLE+2$ so as to work without disabling interrupts and with the current register bank.

Routine EXTRAM

Function Select External Data RAM

Address 0 ECH

Presettings

Postsettings

Input Parameters

Output Parameters

Registers Altered

8048 Control Read

8048 Control Altered

PI

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By Various

EXTRAM

Select External RAM

This routine makes the External Data RAM accessible for read and write operations. The Graphics RAM is thereby inaccessible. The exact effect of the bit masks employed is shown below.

96

X	X	X	X	X	X	X	X
---	---	---	---	---	---	---	---

1	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

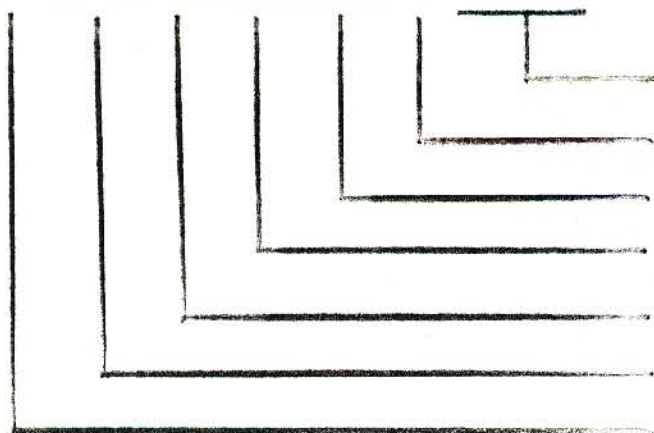
ORL PT1CLR

1	X	1	1	1	1	X	X
---	---	---	---	---	---	---	---

1	0	1	0	1	1	1	1
---	---	---	---	---	---	---	---

ANL ERAM

1	0	1	0	1	1	X	X
---	---	---	---	---	---	---	---



Games ROM unaltered
Keyboard and hand controls disabled
Graphics RAM disabled
External RAM enabled
Hand controls disabled
One-way transfer not selected
Force video high disabled

Routine **EXSETY** (alias **EXTSND**)

Function **Sound Table Handling** (for user-defined sound tables)

Address **04BH**

Presettings The interrupt servicing environment (as set up by **INTSVC**, **VRTETY**)

Postsettings Returns to interrupted code with proper environment

Input Parameters **RBO**: **R1** = first byte of table entry, **R2** = second byte of table entry

Output Parameters **RBO**: **R3** = duration, **R4** = pointer to next sound entry.

Registers Altered **RBO**: **R0**, **R1**, **R2**, **R3**, **R4**

8048 Control Read

8048 Control Altered

Internal RAM Read **MISC (B6)**

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered **A7**, **A8**, **A9**, **AA**.

ROM Data Read

Calls

Called By user

EXSETY (alias EXTEND) Sound Table Handling (user-defined sound tables) 98

These routines are called to step on to the next entry in a sound table. The standard synchronization is with vertical blanking interrupts (frame counts). In this case the user will have got control at location 040AH in the games ROM, having executed INTSVC and VRTETY

For user-defined sound tables, the call is to EXSETY and the user must set
R1 = First byte of table entry
R2 = Second byte of table entry
on calling. This is to be done after the executive call to 40AH e.g.

040A	MOV	A, R4	GET TABLE POINTER
	MOVP	A, @A	GET FIRST BYTE
	MOV	R1, A	PUT IN R1
	INC	R4	INCREMENT TABLE POINTER
	MOV	A, R4	GET TABLE POINTER
	MOVP	A, @A	GET SECOND BYTE
	MOV	R2, A	PUT IN R2
	JMP	EXSETY	GO BACK TO EXECUTIVE

EXSETY takes care of a second increment to R4, if needed.

Actions performed

99

Identifies the function of the new table entry and jumps to the relevant code segment:

TONEPT	new tone
STATUS	sound status change
PAUSE	pause in sound
CHAIN	chaining to a different table
SNBEND	end of table

The new function is set up in the Graphics RAM as appropriate, R3 is reset, the sound control bit of MISC is cleared at the end of the table, and control is returned to the interrupted code.

Routine GEN

100

Function Calculate LSS and move bytes 3 and 4 of a Major Object to a Data Packet

Address 229 H

Presettings Call EXTRAM

Postsettings

Input Parameters ACC = Key code (bits 3-0 only are used)
R0 = output addr in packet R4 = Y-value R6 = colour bits

Output Parameters R5 = LSS-value R6 = colour bits (adjusted)

Registers Altered RBI: R0, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

6244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

LSSCAL

Called By

OUTP

Routine GENI

Function Calculate LSS and Move bytes 3 and 4 of a Major Object to
a Data Packet

Address 22CH

Presettings Call EXTRAM

Postsettings

Input Parameters R0 = output addr in packet R4 = Y-value R5 = key code
R6 = colour bits

Output Parameters R5 = LSS-value R6 = colour bits (adjusted)

Registers Altered R0, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

LSSCAL

SCROUT

GEN and GEN1

Calculate LSS and move bytes
3 and 4 of a Major Object to a Data Packet

GEN can be regarded as an entry point in the GEN1 routine where the caller specifies the key code - of the required major in the ACC. In this case only bits 3-0 are taken, i.e. the routine is intended for digits only.

Where the entry GEN1 is used, the user specifies the key code in RS. All keycodes are valid.

Further parameters are

R0 output address in data packet

R4 Y-value

R6 colour bits

On return R0 contains the address of the next byte to be used in the data packet in the external RAM.

HNDCTL

Hand Control Enable Mask

103

Enable Hand Controls

(No routine in executive; included to show use of masks)

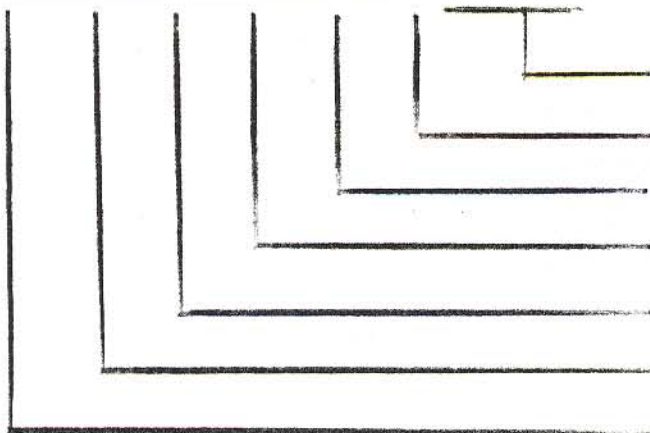
x	x	x	x	x	x	x	x
---	---	---	---	---	---	---	---

1	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

1	x	1	1	1	1	x	x
---	---	---	---	---	---	---	---

1	0	0	1	1	0	1	1
---	---	---	---	---	---	---	---

1	0	0	1	1	0	x	x
---	---	---	---	---	---	---	---



ORL P1, #PT1CLR

ANL P1, #HNDCTL

Games ROM unaltered

Keyboard+hand controls enabled

Graphics RAM disabled

External RAM disabled

Hand controls enabled

One-way transfer not selected

Force video high disabled

Need to restore P1 after reading hand control data.

Routine INTSND

104

Function Sound Table Handling (for ~~executive~~ defined sound tables)

Address 044H

Presettings The interrupt servicing environment (as set up by INTSVC, VRTETY)

Postsettings Returns to interrupted code with proper environment

Input Parameters RBO: R4 = pointer to sound table entry on page 3.

Output Parameters RBO: R3 = duration, R4 = pointer to next sound entry

Registers Altered RBO: R0, R1, R2, R3, R4

8048 Control Read

8048 Control Altered

Internal RAM Read MISC(B6)

Internal RAM Altered

8244 RAM Read

8244 RAM Altered A7, A8, A9, AA

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By user

INTSND

Sound Table Handling (for executive-
defined sound tables). 105

This routine is used to step on to the next entry in an executive-defined sound table. The standard synchronization is with vertical blanking interrupts (frame counts). In this case the user will have got control at location 40AH in the games ROM, having executed INTSVC and VRTETY.

The call is simply `JMP INTSND`. The presettings are correct if INTSVC and VRTETY have been executed.

Actions performed

Identifies the function of the new table entry and jumps to the relevant code segment:

TONEPT	new tone
STATUS	sound status change
PAUSE	pause in sound
CHAIN	chaining to a different table
ENDEND	end of table

The new function is set up in the Graphics RAM as appropriate, R3 is reset, the sound control bit of MISC is cleared at the end of the table, and control is returned to the interrupted code.

Routine : INTSVC

106

Function External Interrupt Service Routine

Address 009H

Presettings

Postsettings PI is restored. Interrupts are reset.

Can also exit via 0406H for vertical blanking handling

Input Parameters

Output Parameters

Registers Altered RBO : R0, R5, R6

8048 Control Read
8048 Control Altered

Internal RAM Read
Internal RAM Altered

8244 RAM Read AI
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

Calls SL8244
Called By user

INTSVC

External Interrupt Service Routine

107
PHILIPS

N.B. The description given below is strictly for INTSVC. The documentation [Ref 2] includes functions of the routine VRTETY, but that routine has no fixed connection with INTSVC.

INTSVC is the standard external interrupt service routine, and can be called at the user's discretion. It is only suitable where the only source of external interrupts is vertical blanking. This is because control is only given to the games ROM on vertical blanking interrupts; in other cases it just returns to the interrupted code. It is not possible ~~to~~ for the user to pre-process for other interrupts as INTSVC would then serve no purpose and also because Graphics RAM location 0A1H would then be read twice, once by the user and once by INTSVC. Reading that location clears the interrupt request, so it is important to identify all interrupting sources and process them all after just one access to 0A1H.

CALLING SEQUENCE

0402 JMP INTSVC

ACTIONS PERFORMED

saves ACC and PI in R5 and R6 of MBO
checks for a vertical blanking interrupt by reading 0A1H
if found, a jump is made to 406H
if not found, restores the environment of before the interrupt and returns to the interrupted code.
The label RESTOR can be used to jump to by the vertical blanking routine.

In practice the vertical blanking routine will be VRTETY (q.v.).

Routine KEYBRD

Function Scan keyboard for key entered

Address 0B0H

Presettings

Postsettings

Input Parameters RBO: R7 = key code obtained in previous call

Output Parameters RBO: R7 = new key code or "bad data" = 0C0H

Registers Altered RBO: R0, R1, R2, R6

8048 Control Read

8048 Control Altered P2

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By KEYCHK

KEYBRD

Scan keyboard for key entered.

109

This is the standard way to test for keyed input. The routine rejects transient phenomena caused by (de) bouncing. The routine disables interrupts so as to be able to use register bank 0.

This is a little unfortunate since it means that interrupt servicing can be delayed by nearly 800 μ sec! It is therefore recommended to call this routine while the 8244/8245 is busy drawing objects, so that there is no chance of a vertical blanking interrupt and no need for horizontal blanking interrupts. Sound is not so critical. The routine re-enables interrupts on return.

(e.g. KEYBRD+2, KEYCK)

Alternative entry points $\swarrow$ could be used to work with interrupts enabled and register bank 1. In this case the caller must take care to make the right presettings and to allow for loss of the user's registers.

The routine returns the code of the key currently depressed.

Since the keyboard does not cause interrupts, it is necessary

to poll the keyboard regularly. In this way a released key is detected, and the code for this is saved. The routine indicates whether a release or different key preceded the currently pressed key. For keycodes, see Section 5.

Calling sequence : `CALL KEYBRD`

On Return

R7 contains : `OFFH` indicating that no key is being pressed

or `keycode` indicating that the given key is being pressed and that a release or different key preceded it.

or `keycode + 00H` indicating either a transient effect (bounce) or that the key is being depressed but that this is the same state as the previous call to KEYBRD.

Routine KEYCHK

Function Wait for keyed input and make beep sound.

Address 13DH

Presettings R0: R7 = key code of previous scan of keyboard

Postsettings

Input Parameters

Output Parameters ACC = key code of depressed key

Registers Altered R0: R0, R1, R2, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls KEYBRD, SNDSET
Called By START

KEYCHK

Wait for keyed input and make beep sound 111

This routine waits until a key has been depressed. It does this by scanning the keyboard after each vertical ^{blanking} interrupt. It is not possible to do anything else at main program level while this routine is busy. When valid key data has been obtained, the routine makes a beep and returns the keycode in the Acc. For keycodes, refer to section 5.

Routine LDMAJ

Function Calculate LSS of a Major Object and load into
Graphics RAM

Address 261H

Presettings Call 5L8244, DISABL

Postsettings

Input Parameters R0 = 3rd byte address of major R4 = Y-value
R5 = Key code R6 = colour attributes

Output Parameters R5 = LSS-value R6 = colour attributes (adjusted)

Registers Altered R0, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

Bytes 3 and 4 of major object

ROM Data Read

Calls

Called By

LDMAJ

Calculate LSS of a Major Object
and load into Graphics RAM

113

This routine performs the LSSCAL functions (determines the LSS value given the Y-coordinate and key code and colour bits) and writes the two LSS-bytes to the Graphics RAM using R0 as a pointer. This routine is useful for modifying existing majors in data packets, and with ^{subordinate} group majors where the first two bytes are irrelevant.

on entry R0 →

on exit R0 →

X
Y
LSS
colour + LSS9

unaffected
unaffected
adjusted
adjusted

Routine LSSCAL

Function Calculate LSS-value of Major Object

Address 14BH

Presettings

Postsettings

Input Parameters RBI: R4 = Y-value R5 = Key code R6 = colour attr

Output Parameters RBI: R5 = LSS-value R6 = Colour attr (adjusted).

Registers Altered RBI: R5, R6

8048 Control Read

8048 Control Altered C

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

LDMAJ

LSSCAL

Calculate LSS -value of Major Object 1/15

This routine is useful when major objects move in the Y-direction, since the LSS (= linear select store) value then changes. The LSS value is $(8 \times \text{keycode}) + (Y/2)$. It is not always necessary to call LSSCAL for changes in Y; refer to section 4 for some hints about this.

The definition of a Major Object in the Graphics RAM is

Y	Y	Y	Y	Y	Y	Y	Y
X	X	X	X	X	X	X	X
LSS	LSS	LSS	LSS	LSS	LSS	LSS	LSS
X	X	X	X	B	G	R	LSS9

Blue/Green/Red/msb of LSS-value

The input parameters to LSSCAL are the Y value, the keycode and the fourth byte as in the above definition, but without the LSS9 setting. The output values are the third byte as above (LSS-value) and the adjusted fourth byte (i.e. with the LSS9 setting).

Routine MULT

Function Multiply

Address 3DDH

Presettings

Postsettings

Input Parameters $R2 = \text{multiplier}$ $R3 = \text{multiplicand}$

Output Parameters $ACC = \text{product (modulo 256)}$

Registers Altered $R2$

8048 Control Read
8048 Control Altered

Internal RAM Read
Internal RAM Altered

8244 RAM Read
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

Calls
Called By

MULT

Multiply

117

This routine performs unsigned multiplication modulo 256 by repeated addition of the multiplicand. The calling parameters are

R2 := multiplier

R3 := multiplicand

If possible one should assign the multiplicand to the higher of the two values; this works faster. There is no carry indication.

Comment

The routine unnecessarily separately tests for zero multiplier and zero multiplicand. The routine would work using MULT1 as the entry point, saving 8 bytes in the executive.

Routine **NXTINT**

Function Wait for next vertical blanking interrupt.

Address 176H

Presettings

Postsettings

Input Parameters

Output Parameters

Registers Altered

8048 Control Read

8048 Control Altered

FI

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

NXTINT

Wait for next interrupt

119

This routine can be called in a main program when there is no more processing to do for the current video frame cycle. Control is returned to the caller after processing the vertical blanking interrupt.

NXTINT can be regarded as the idle loop in a games program. The routine simply loops, testing whether Flag FI has been set. FI is set before the return of the vertical blanking interrupt routine. When the interrupt routine returns to the interrupted code (NXTINT), the change in FI is detected and control returns to the caller of NXTINT.

Routine OPMJ

Function Calculate LSS of a Major Object and load entire Major into Graphics RAM

Address 3EA

Presettings select Graphics RAM (528244), call DISABL

Postsettings

Input Parameters R0 = Addr of Major, R3 = X-value, R4 = Y-value
R5 = Key Code, R6 = Colour attributes.

Output Parameters R5 = LSS-value R6 = colour attributes (adjusted)

Registers Altered R0, R3

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls LDMAT
Called By

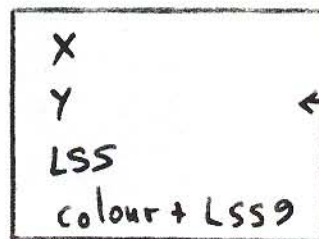
OPMJ

121
Calculate LSS of a Major Object
and load entire Major into Graphics RAM.

Note. This routine has a bug in it and cannot be used for master group major objects.

The routine outputs a major object to the Graphics RAM. The routine examines the destination address and in the case of group major objects other than 1, 5, 9 and 13 it skips writing the x-coordinate. Unfortunately the above are subordinate group major objects, not the masters, (which are 4, 8, 12, 16). For single major objects there is no problem.

Memory Diagram



← only written for group majors
1, 5, 9 and 13
and single majors.

The bug is particularly unfortunate as the check for subordinate group major objects is unnecessary — it does no harm to insert the Y coordinate and it is quicker than to test and avoid it.

Routine OTFGST

Function Output Flag Set and Terminate Data Packet

Address 132H

Presettings Select External RAM (EXTRAM)

Postsettings 8244 selected

Input Parameters RBI: R0 = pointer to next byte after output packet in descending order.

Output Parameters

Registers Altered RBI: R0

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered MISC(87)

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered Zero is written to location pointed to by R0.

ROM Data Read

Calls SL8244

Called By SCORE, OUTP, TIMER

OTFGST

Output Flag Set

123

This routine takes as parameter in R0 the address of a byte in External Ram to be zeroed, indicating the end of a data packet for the OUTPUT routine. The routine then sets bit 7 of the MISC byte in internal RAM to indicate that output is active. The routine finally selects the Graphics RAM (via a call to 5C8244) and returns.

Note that on entry the External RAM must have been selected.

If the OUTPUT routine is not to be called immediately, but only on vertical blanking interrupts, it may be necessary to reselect External RAM on regaining control from the routine.

Routine OUTPUT

Function Move Data Packets From EXT. RAM to Graphics RAM.

Address 089H

Presettings Disable Video , Set up Packet Data starting at Ext Ram 7FH

Postsettings 8244 selected, video ^{en}disabled

Input Parameters

Output Parameters

Registers Altered

8048 Control Read
8048 Control Altered P1

Internal RAM Read
Internal RAM Altered

8244 RAM Read
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

Calls
Called By VRTETY

Routine OUTF

Function Output Time

Address IFCH

Presettings Call EXTRAM

Postsettings

Input Parameters R4 = Y-value, R6 = colour attributes

Output Parameters

Registers Altered R0, R1, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered Data Packet at 7F to 73

ROM Data Read

Calls

Called By GEN
TIMER

OUTP

Output Time

126

This routine converts the time as held in **MINES** and **SECNDS** (locations 1 and 2 of external RAM) into a data packet to be sent to the Graphics RAM during the next vertical interrupt processing.

The data packet is built from address 7F downwards. This is the same ^{address} as used by the routine SCORE! Beware of a clash.

Structure of the data packet

7F	2	
7E	42 H	address of group maj
7D	.	(10's) minutes LSS
7C	2	
7B	52 H	
7A	.	(units) minutes LSS
79	2	
78	56 H	
77	.	(10's) seconds LSS
76	2	
75	4A H	
74	.	(units) seconds LSS
73	0	

Routine OUTPT1

Function Move Data Packets from EXT. RAM. to Graphics RAM.

Address 08BH

Prerequisites Disable Video, Set up Packet Data anywhere in EXT. RAM.

Postconditions 8244 Selected, video disabled

Input Parameters

Current Register Bank: R0 = address of data packet in EXT. RAM

Output Parameters

Registers Altered

Current Register Bank: R0, R1, R2

8048 Control Read

8048 Control Altered P1

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

user

OUTPUT
and OUTPT1

Move Data Packets from External RAM
to Graphics RAM

128

This routine is called by VRTETY (if the control bit of MISC indicates this) and can be called by the user at any other time providing:

- The 8244/8245 chip has been selected (Graphics RAM)
- it is a suitable stage in the frame scan to allow OUTPUT to disable video and the grid to make data changes to the Graphics RAM.

On return the 8244/8245 is selected and the display is reenabled.

Data Packet Format

Ext RAM

↓
decreasing
memory
addresses!

7F
7E

BYTE COUNT (NO. OF DATA BYTES)
GRAPHICS RAM ADDRESS
DATA BYTE 1
DATA BYTE
BYTE COUNT GRAPHICS RAM
DATA BYTE 1
DATA BYTE N
ZERO

called OUTADS

First packet

Second packet

3rd data packet

The entry point OUTPT1 is used when the data packets are pointed to by R0 instead of being fixed at location 7F (Note that VRTETY only knows about OUTPUT).

Routine RDBT

Function Reset (=clear) a bit in External RAM or Graphics RAM

Address 280 H

Presettings Select External RAM or Graphics RAM (EXTRAM/SL8244)

Postsettings

Input Parameters ACC = bit number
R1 = address of byte

Output Parameters ACC = 1 R2 = bit mask as used to AND in the required bit

Registers Altered R1, R2

8048 Control Read
8048 Control Altered

Internal RAM Read
Internal RAM Altered

8244 RAM Read
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

Calls TDBT
Called By

RDBT

Reset (= Clear) a bit.

130

This routine clears a bit at a given offset from an address. The routine works with External RAM or Graphics RAM depending on which has been selected. It cannot be used for internal RAM.

The bit numbering is nonstandard. Refer to the description of TDBT for this.

Routine RESLV

Function Get status of joystick

Address 38FH

Presettings

Postsettings

Input Parameters R1 = hand control number (0 or 1)

Output Parameters R1 = raw hand control data, R2 = delta X, R3 = delta Y,
F0 = fire button.

Registers Altered R2, R3

8048 Control Read P1, BUS

8048 Control Altered F0, P2,

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

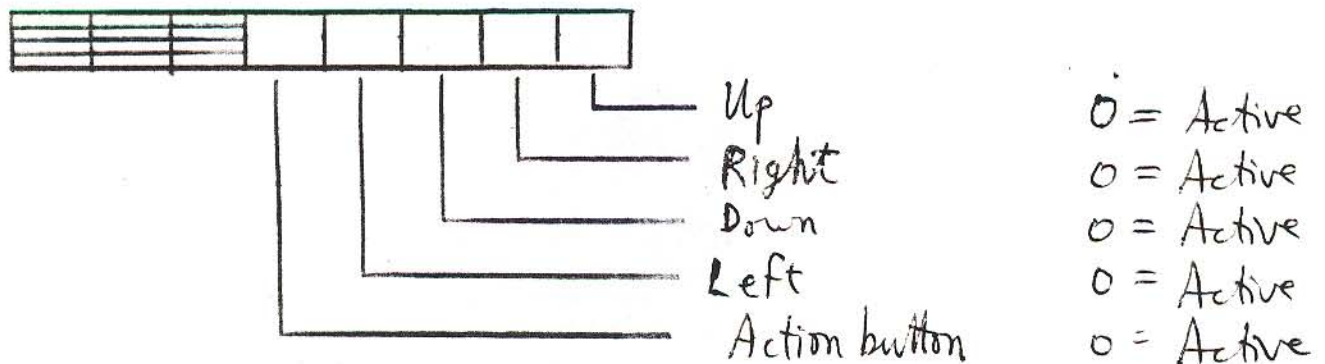
RESLV

Get status of Joystick.

132 01111003

This routine enables the hand controls via port 1, selects the required joystick, 0 or 1, via port 2, and reads the joystick data from the BUS.

The BUS data is the raw hand control data and has the following content



This raw data is returned in R1

Refined data is also returned in R2, R3 and F0

R2 = -1, 0, or 1

R3 = -1, 0, or 1

F0 = 0 or 1

depending on the X attitude of the joystick (-1 = left, +1 = right)
depending on the Y attitude of the joystick (-1 = up +1 = down)
(0 = action button not pressed, 1 = action button pressed)

Routine RNDM

133

Function Random Number Generator

Address 2A4H

Presettings

Postsettings

Input Parameters R5=seed (0 to F)
R7=-(max value +1) of returned nibble

Output Parameters R3 and ACC = resulting value, right justified..

Registers Altered R3, R5

8048 Control Read
8048 Control Altered

Internal RAM Read
Internal RAM Altered

8244 RAM Read
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

Calls
Called By RNDM2

RNDM

random number generator

134

This routine returns a random number in the range 0 to N . N can be in the range 1 to 15 and is implied by setting $R7$ to $-(N+1)$ in the calling sequence. $R5$ must be supplied as a seed (0 to 15) and the returned value in $R5$ can be used as the next seed. The routine ^{then} has a cycle of N random numbers, in which each number occurs just once. This is rather poor.

The method is simply to cyclicly access the entry in a table of 16 random nibbles. Where $N < 15$, rejection of out-of-range values takes place by returning to the start of the routine.

Routine RNDM2

Function Random Number Generator using the Timer

Address 293H

Presettings

Postsettings

Input Parameters R2 = --(max value) for most significant nibble
R7 = --(max value) for least significant nibble

Output Parameters R3 and ACC = most significant nibble, right justified (!)
R2 = least significant nibble, right justified

Registers Altered R2, R3, R4, R5, R7

8048 Control Read Timer

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls RNDM
Called By

RNDM2

Random Number Generator using the Timer

136

This routine returns two random nibbles. The range of each nibble can be separately determined.

The routine works by making two calls to RNDM using seeds based on the two nibbles of the timer value. If the timer is not in use, or only uses small counts, then RNDM2 will not produce proper random numbers.

The following call obtains two random nibbles, one in the range 0-7 and the other in the range 0-F

```
MOV R2, #0F8H
MOV R7, #0F0H
CALL RNDM2
```

lsb R2 = random number
in the range 0 to 7

msb R3 and ACC = random number
in the range 0 to 15

Both the lsb and msb are right-justified.

Comment

The routine rather blatantly squanders three bytes by the sequence

```
CALL RNDM
RET
```

RNDM:

The above instructions are unnecessary.

Routine SCORE

Function Output Score Digits

Address 17CH

Presettings Output flag must be cleared (MISC-byte)

Postsettings 8244 selected

Input Parameters ACC = BCD Score Value R1 = address in Graphics RAM for majors
R3 = X-position R4 = Y-position R6 = colour bits

Output Parameters R3 = X-position + 8

Registers Altered RBI: R0, R2, R3, R5

8048 Control Read

8048 Control Altered P1

Internal RAM Read

Internal RAM Altered MISC (B7)

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered Bytes 7F to 75 written

ROM Data Read

Calls

Called By

EXTRAM,

OTFGST, SCROUT

SCORE

Output Score Digits

138

This routine outputs two decimal digits (typically a score) at a position specified by the caller, in a colour specified by the caller. This is done using two consecutive majors, whose address in the Graphics RAM is also specified by the caller.

The output does not take place immediately. A data packet is prepared in locations 7F to 76 of the External RAM, to be output on the next vertical blanking interrupt by the executive. It is not possible to vary the address in External RAM at which the data packet is constructed. The structure is

7F	8
7E	ADDR. OF MAJOR
7D	Y co-ord
7C	X co-ord
7B	LSS value
7A	Colour + LSS9
79	Y-co-ord.
78	X-co-ord
77	LSS value
76	Colour + LSS9
75	0
74	

Note that the description in the executive listing and in Reference 2 state (incorrectly) that the last location used is 76.

Routine SCROUT

139

Function Calculate LSS and Output Major Object to Data Packet

Address 197H

Presettings Call EXTRAM

Postsettings

Input Parameters R0 = Output Addr. in Packet, R3 = X-value, R4 = Y-value
R5 = Key Code, R6 = colour bits.

Output Parameters R5 = LSS, R6 = adjusted colour bits

Registers Altered R0, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

GENI

SCORE

SCROUT

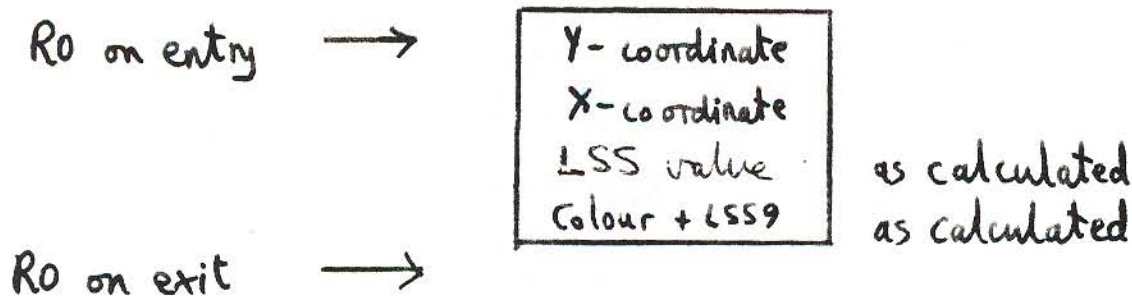
Calculate LSS and output
Major Object to a Data Packet

140
211128

This routine constructs the description of a Major Object in a data packet in External RAM. The caller specifies the address in External RAM, the X and Y co-ordinates, the key code and the colour bits.

On return R0 contains the address of the next free byte to be used in the data packet in the external RAM.

Memory Diagram



Routine SDBT

Function Set a bit in External RAM or Graphics RAM

Address 28AH

Presettings Select External RAM or Graphics RAM (EXTRAM/SL8244)

Postsettings

Input Parameters ACC = bit number
R1 = address of byte

Output Parameters ACC = 0 R2 = bit mask as used to AND in the required bit

Registers Altered R1, R2

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By

SDBT

Set (to 1) a bit

142

This routine sets a bit at a given offset from an address. The routine works with External RAM or Graphics RAM depending on which has been selected. It cannot be used for internal RAM.

The bit numbering is nonstandard. Refer to the description of TDBT for this.

Routine SL 8244

143

Function Select Graphics RAM

Address 0E7H

Presettings

Postsettings

Input Parameters

Output Parameters

Registers Altered

8048 Control Read

8048 Control Altered

P1

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By Various

SL8244

Select 8244/8245

144

This routine makes the Graphics RAM accessible for read and write operations. The External Data RAM is thereby inaccessible. The exact effect of the bit masks employed is shown below:

The mask is the same as VDLO.

X	X	X	X	X	X	X	X
---	---	---	---	---	---	---	---

1	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

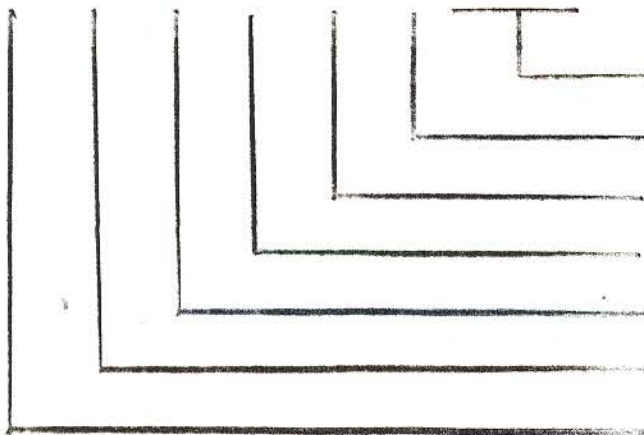
ORL PT1CLR

1	X	1	1	1	1	X	X
---	---	---	---	---	---	---	---

1	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---

ANL \$8244

1	0	1	1	0	1	X	X
---	---	---	---	---	---	---	---



Games ROM unaltered
Keyboard + hand controls disabled
Graphics RAM enabled
External RAM disabled
Hand controls disabled
One-way transfer not selected
Force video high disabled

Routine SNDSET

Function Initialize sound table processing

Address 1A2H

Presettings

Postsettings Interrupts Enabled

Input Parameters ACC = address of sound table

Output Parameters

Registers Altered RBO: R3, R4

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered MISC (B6)

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By START, KEYCHK

This routine sets up the necessary parameters for the executive to start (and later work through) an executive-resident sound table. This is done by setting the sound bit of the MSC byte in internal RAM and by setting the sound table pointer to the sound table address, and the sound counter to 1 so that it expires on the next vertical blanking interrupt.

The routine can also be used to set up sound from a user-defined table. In this case special attention must be paid to sound processing on vertical blanking interrupts (q.v. in this Section).

Notes

The comment in the executive listing that the SNDSET routine initializes sound interrupts is misleading. The executive does not work with sound interrupts, Vertical blanking interrupts determine The duration of sound effects.

Routine START

Function Start of game selection and initialization sequence

Address 2C3H

Presettings RBO selected

Postsettings 8244 selected (Graphics RAM). Exit is to 408H via JMP.

Input Parameters

Output Parameters ACC = key code of depressed key

Registers Altered R0, R1, R2, R3, R4, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered Various (for majors and sound and control)

ROM Data Read

Calls
Called By CLEAR, DISABL, OPMJ, ENABLE, SNDSET, KEYCHK, CLRMAJ.

START

Start of game selection and
initialization sequence.

148

This routine clears all ^{free} internal RAM, external RAM and graphics RAM, then outputs

SELECT GAME

using 11 single major objects. These appear in various colours. A sound table is initiated to produce the start up sound. The keyboard is read (using routine KEYCHK) to get the game select code. At this point the SELECT GAME display is cleared and a jump is made to 408H with the key code in the ACC.

Note. The routine is JUMPed to and not CALLED.

Routine TDBT

149

Function Test a bit in External RAM or Graphics RAM

Address 26AH

Presettings Select External RAM or Graphics RAM (EXTRAM/SL8244)

Postsettings

Input Parameters ACC = bit number
R1 = Address of byte

Output Parameters ACC = bit requested, all other bits zero.
R2 = bit mask as used to AND in the required bit.

Registers Altered R1, R2.

8048 Control Read
8048 Control Altered

Internal RAM Read
Internal RAM Altered

244 RAM Read
8244 RAM Altered

External RAM Read
External RAM Altered

ROM Data Read

Calls
Called By RDBT, SDBT

TDBT

Test a bit

150

The routine can be used to get a bit at a given offset from an address. One can imagine a memory block as an array of logical (or Boolean) variables.

The bit numbering is nonstandard: The example below illustrates this.

Example:

Bit numbering as used by TDBT, RDBT and SDBT with respect to byte 6

2	32	31	30	29	28	27	26	25
3	24	23	22	21	20	19	18	17
4	16	15	14	13	12	11	10	9
5	8	7	6	5	4	3	2	1
6								

EG	0							
----	---	--	--	--	--	--	--	--

The routine work with external RAM (or Graphics RAM, depending on which has been selected). It cannot be used for internal RAM.

Calling Sequence : ACC := bit number (above convention)
RI := address (above convention)

Return : ACC = bit, in place, remaining bits zero
RI := address referenced.

N.B. The description in Ref 2 is inaccurate regarding the returned value of RI.

Routine **TIMER**

Function Maintain the time on screen.

Address 1B0H

Presettings Call TIMINT. Set MINUTES and SECNDS. Fill digits
in Graphics RAM

Postsettings 8244 selected

Input Parameters R4 = Y-value R6 = colour bits

Output Parameters

Registers Altered

8048 Control Read

8048 Control Altered RBI: R0, R1, R2, R5, R6

Internal RAM Read

Internal RAM Altered MODU60

8244 RAM Read

8244 RAM Altered

External RAM Read

External RAM Altered

01, 02 for minutes and seconds.

ROM Data Read

Calls

Called By

GEN, WORD, LSSCAL, O TFGST

TIMER

Maintain The time on Screen.

152

This routine maintains the time in the form

MM:SS

(minutes and seconds).

on the screen using

The first two group majors.

The timer must have been initialized by the routine TIMINT.

The routine has nothing to do with the 8048 internal timer.

The timer runs when the MODU60 counter = 59. This counter is held as a binary number. Bit 6 of MODU60 indicates whether

the TIMER should be incremented (bit set) or decremented (bit cleared). Bit 7, if set, indicates that the timer has stopped and no further action will be taken.

TIMER, when called and when the above conditions are fulfilled, increments or decrements SECNDS and MINTES in external RAM and prepares an output packet using OUTP(q.v.).

Routine TIMINT

152

Function Initialize Timer

Address 23AH

Presettings Call SL8244, DISABL

Postsettings Set initial digits in 8244 (¹⁻⁸⁻⁸⁻⁸MIN, SEC at 46H, 4EH, 5AH, 5EH)

Input Parameters R3 = X-value R4 = Y-value R6 = colour bits

Output Parameters R6 = colour bits (adjusted)

Registers Altered R0, R1, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

Internal RAM Altered

8244 RAM Read

8244 RAM Altered Group Majors 1 and 2.

External RAM Read

External RAM Altered

ROM Data Read

Calls

Called By LDMAJ

TIMINT

Initialize timer

154

For a description of the timer display, see the routine TIMER.
TIMINT sets up the following group major objects:

a	b	c	d	e	f	g	h
		:			blank	blank	blank

The time digits themselves are undefined and must be loaded by the user.

The X and Y coordinates and the colour are determined by the caller. Routine TIMER maintains the time.

g_1 | a | c | e | g |

g_2 | b | d | f | h |

Routine VRTETY

Function Vertical Blanking Interrupt Service Routine

Address 01AH

Presettings RBO selected, 8244 selected, ACC, PI saved.
(all as done by INTSVC)

Postsettings Restores environment for and returns to interrupted code
Can also exit to 040AH for user sound handling. FI set

Input Parameters

Output Parameters

Registers Altered RBO: R0, R1, R2, R3, R4, R5, R6

8048 Control Read

8048 Control Altered

Internal RAM Read

MISC (B6)

Internal RAM Altered

MISC (B7), MODU60 (B5-0), DVSTAT

8244 RAM Read

A2

8244 RAM Altered

Data, A0

External RAM Read

Data Packets (if present)

External RAM Altered

ROM Data Read

Sound Tables if sound active

Calls

OUTPUT

Called By

user

VRTETYVertical Blanking Interrupt Service Routine

VRTETY performs standard actions which are synchronized to vertical blanking interrupts. These are: update counter, move data packets to the Graphics RAM and service sound.

CALLING SEQUENCE: 0406 JMP VRTETY
 OR
 0406 JMP user-extra-actions

 user-extra-actions
 JMP VRTETY

ACTIONS PERFORMED

- Flag FI is set so as to hand control out of the NXTINT loop, should the main program be there.
- The overlap status byte is saved in data RAM. This is necessary since it will shortly be cleared by the 8244-8245 for the next frame scan.
- The modulo 60 counter, bits 5-0 are incremented and cleared if 60 is reached.
- If the miscellaneous flag MISC bit 7 is set, it is cleared and OUTPUT (q.v.) is called, a routine for transferring data from the external RAM to the Graphics RAM.
- If the miscellaneous flag MISC bit 6 is set, then sound is processed. The frame count for the current tone is decremented (register R3). If this has expired, a jump is made to 40AH so that the user can call the right sound service routine.
- If no sound servicing is required, a jump is made to RESTOR (see routine INTSVC) where a return is made to the interrupted code.

Section 9 Standard Techniques

157

Contents Programming Techniques

Game Techniques

Indexing from a current page data table using an index in internal RAM

```
MOV R0, #index
MOV A, @R0          GET INDEX FROM INTERNAL RAM
ADD A, #low.dtab
MOVP A, @A
```

DTAB 030H, 050H, 040H

Switch-statement using a parameter in internal RAM

```
MOV R0, #index
MOV A, @R0
ADD A, #low.jtab
JMPP @A
```

JTAB .low.case1, .low.case2, .low.case3

For-loop

```
MOV R2, #N
LOOP  loop actions
      DJNZ R2, LOOP
```

Subtraction from number in ACC

CPL A
 ADD A, — (various formats)
 CPL A

Optimization in a subroutine

Where the return follows a subroutine action (RET-RETR as required), replace

CALL SUBRX
 RET(R)

by

JMP SUBRX

or attempt to place SUBRX at the location of the above JMP, making the JMP altogether redundant.

Action in a register (e.g. shift left)

XCH A, R5

RL A

XCH A, R5

Action on a byte in internal RAM indexed via R0 (e.g. shift left)

XCH A, @R0

RL A

XCH A, @R0

SYSTEM EQUATES (i) macro SMKEYS (keycodes)

SQN0	EQU	000H	numeric
SQN1	EQU	001H	
SQN2	EQU	002H	
SQN3	EQU	003H	
SQN4	EQU	004H	
SQN5	EQU	005H	
SQN6	EQU	006H	
SQN7	EQU	007H	
SQN8	EQU	008H	
SQN9	EQU	009H	
SQ10	EQU	030H	

SQAA	EQU	020H	alphabetic
SQAB	EQU	025H	
SQAC	EQU	023H	
SQAD	EQU	01AH	
SQAE	EQU	012H	
SQAF	EQU	01BH	
SQAG	EQU	01CH	
SQAH	EQU	01DH	
SQAI	EQU	016H	
SQAJ	EQU	01EH	
SQAK	EQU	01FH	
SQAL	EQU	00EH	
SQAM	EQU	026H	
SQAN	EQU	02DH	
SQAO	EQU	017H	
SQAP	EQU	00FH	
SQAQ	EQU	018H	
SQAR	EQU	013H	
SQAS	EQU	019H	
SQAT	EQU	014H	
SQAU	EQU	015H	
SQAV	EQU	024H	
SQAW	EQU	011H	
SQAX	EQU	022H	
SQAY	EQU	02CH	
SQAZ	EQU	021H	

SQSP	EQU	00CH	space
SQEN	EQU	02FH	full matrix = enter
SQPL	EQU	010H	+ plus
SQMI	EQU	028H	- minus
SQDV	EQU	02AH	÷ times
SQMU	EQU	029H	x times
SREQ	EQU	02BH	= equals
SQRM	EQU	00DH	? question mark
SQCO	EQU	00AH	: colon
SQFS	EQU	027H	. full stop
SQDR	EQU	00BH	\$ dollar

SQMS	EQU	03AH	man stationary
SQML	EQU	035H	man left stoppe
SQLL	EQU	034H	man left runni
SQMR	EQU	032H	man right stoppe
SQRR	EQU	033H	man right runni
SQNE	EQU	02EH	/ diag N-ea
SQNW	EQU	03BH	\ diag N-we
SQRA	EQU	036H	→ right arrow
SQTR	EQU	037H	↑ tree
SQFR	EQU	02FH	● fragment
SQBT	EQU	03FH	⚓ boat
SQPN	EQU	03DH	✈ plane
SQSM	EQU	03CH	⚓ submarine
SQTK	EQU	03EH	⚓ tank
SQBR	EQU	038H	⚓ bottom right
SQBL	EQU	039H	⚓ bottom left

SIMILAR MACRO FOR
DIRECT USE OF
8 * KEYCODE ?
[used in shadow room
major transfer]

SYSTEM EQUATES (2) macro SMCOLS

colours and attributes

SQMIBU EQU 020H
 SQMIGR EQU 010H
 SQMIRE EQU 008H
 SQMIBK EQU 000H
 SQMIYE EQU SQMIRE + SQMIGR
 SQMITU EQU SQMIBU + SQMIGR
 SQMIPU EQU SQMIRE + SQMIBU
 SQMIWH EQU 038H

minor blue
 minor green
 minor red
 minor black
 minor yellow
 minor turquoise
 minor purple
 minor white

SQMIDB EQU 004H
 SQMISM EQU 002H
 SQMIX9 EQU 001H

double size
 smoothing
 X9 half pixel

SQMABU EQU 008H
 SQMAGR EQU 004H
 SQMARE EQU 002H
 SQMABK EQU 000H
 SQMAYE EQU SQMARE + SQMAGR
 SQMATU EQU SQMABU + SQMAGR
 SQMAPU EQU SQMARE + SQMABU
 SQMAWH EQU 00EH

major blue
 major green
 major red
 major black
 major yellow
 major turquoise
 major purple
 major white

SQBGBU EQU 008H
 SQBGR EQU 010H
 SQBGRE EQU 020H
 SQBG BK EQU 000H
 SQBG YE EQU SQBGRE + SQBGR
 SQBG TU EQU SQBGBU + SQBGR
 SQBG PU EQU SQBGRE + SQBGBU
 SQBG WH EQU 038H

background blue
 background green
 background red
 background black
 background yellow
 background turquoise
 background purple
 background white

SQGRBU EQU 001H
 SQGRGR EQU 002H
 SQGRRE EQU 004H
 SQGRBK EQU 000H
 SQGRYE EQU SQGRRE + SQGRGR
 SQGR TU EQU SQGRBU + SQGRGR
 SQGR PU EQU SQGRRE + SQGRBU
 SQGR WH EQU 007H

grid blue
 grid green
 grid red
 grid black
 grid yellow
 grid turquoise
 grid purple
 grid white

SQGRBR EQU 040H

grid bright

SYSTEM EQUATES (3)

macro

SM.VRAM

video ram addresses

SVMIN1	EQU	000H
SVMIN2	EQU	004H
SVMIN3	EQU	008H
SVMIN4	EQU	00CH

SVMAJ1	EQU	010H
SVMAJ2	EQU	014H
SVMAJ3	EQU	018H
SVMAJ4	EQU	01CH
SVMAJ5	EQU	020H
SVMAJ6	EQU	024H
SVMAJ7	EQU	028H
SVMAJ8	EQU	02CH
SVMAJ9	EQU	030H
SVMAJA	EQU	034H
SVMAJB	EQU	038H
SVMAJC	EQU	03CH

SVGP11	EQU	040H
SVGP12	EQU	044H
SVGP13	EQU	048H
SVGP14	EQU	04CH
SVGP21	EQU	050H
SVGP22	EQU	054H
SVGP23	EQU	058H
SVGP24	EQU	05CH
SVGP31	EQU	060H
SVGP32	EQU	064H
SVGP33	EQU	068H
SVGP34	EQU	06CH
SVGP41	EQU	070H
SVGP42	EQU	074H
SVGP43	EQU	078H
SVGP44	EQU	07CH

SVMP1	EQU	080H
SVMP2	EQU	088H
SVMP3	EQU	090H
SVMP4	EQU	098H

SVCTRL	EQU	0A0H
SVSTAT	EQU	0A1H
SVOVST	EQU	0A2H
SVCLR	EQU	0A3H
SVSTR	EQU	0A4H
SVXSTR	EQU	0A5H

SVSDSR	EQU	0A7H
SVSDCT	EQU	0AAH

SVGR1H	EQU	0C0H
SVGR1B	EQU	0D0H
SVGR1V	EQU	0E0H

SYSTEM EQUATES (4)

macro SM.VREG

video ram register content equates
except sound : see further on.

SQVEHI	EQU	001H	for SVCTRL (= 0A0H)	enable horizontal interrupt
SQVFPS	EQU	002H		forced position stroke
SQVESI	EQU	004H		enable sound interrupt
SQVEGR	EQU	008H		enable grid
SQVEEO	EQU	010H		enable external overlap
SQVEDP	EQU	020H		enable display
SQVDEN	EQU	040H		dot enable
SQVGRW	EQU	080H		grid wide

SQVHST	EQU	001H	for SVSTAT (= 0A1H)	horizontal status
SQVPSS	EQU	002H		position stroke status
SQVSNs	EQU	004H		sound needs service
SQVVST	EQU	008H		vertical status
SQVEXO	EQU	040H		external overlap
SQVMJO	EQU	080H		major overlap

SYSTEM EQUATES (S)

macro

SMEXEC

executive entries

CALHD	EQU	3B1H
CLEAR	EQU	0F1H
CLRMAT	EQU	16BH
DISABL	EQU	11CH
DVDE	EQU	3CFH
ENABLE	EQU	127H
EXTRAM	EQU	0ECH
EXSETY	EQU	04BH
GEN	EQU	229H
GEM	EQU	22CH
INTSND	EQU	044H
INTSVC	EQU	009H
KEYBRD	EQU	08DH
KEYCHK	EQU	13DH
LDMAJ	EQU	261H
LSSCAL	EQU	14BH
MULT	EQU	3DDH
NXTINT	EQU	176H
OPMJ	EQU	3EAH
OTFGST	EQU	132H
OUTPUT	EQU	089H
OUTP	EQU	1FCH
OUTPI	EQU	08BH
RESLV	EQU	38FH
RESTOR	EQU	014H
SCORE	EQU	17CH
SCROUT	EQU	197H
SLB244	EQU	0E7H
SNDSET	EQU	1A2H
START	EQU	2C3H
TIMER	EQU	18DH
VRTETY	EQU	01AH

TONE2	EQU	328H
STPSND	EQU	32CH
CRASH	EQU	32EH
TONE1	EQU	33CH
SCLUP	EQU	34AH
TONE4	EQU	356H
TONES	EQU	35AH
SCLDN	EQU	35EH
GUNHT	EQU	36AH

SYSTEM EQUATES (6)

macro SMPORT

port settings and masks

SQPBT1 EQU 07CH

SQPBT2 EQU 0E7H

SQPCR EQU 0BCH

SQPXR EQU 0AFH

SQPVR EQU 0B7H

SQPKYB EQU 0BBH

SQPHCT EQU 09BH

block transfer (=1 way) ANL-mask for PI

block transfer (=1 way) ORL-mask for PI

clear select functions ANL-mask for PI

select EXT RAM ORL-mask for PI

select VIDEO RAM ORL-mask for PI

select KEYBOARD ANL-mask for PI

select HAND CONTROL ANL-mask for PI

SYSTEM EQUATES (7) macro SMSOUN

sound functions

SQSD01	EQU	01	duration	SQSTAT	EQU	040H	status
SQSD02	EQU	02		SQSENB	EQU	080H	enable
SQSD03	EQU	03		SQSEBT	EQU	040H	unused bit
SQSD04	EQU	04		SQS4KS	EQU	020H	4K-shifting
SQSD05	EQU	05		SQS1KS	EQU	000H	2K-shifting
SQSD06	EQU	06		SQSN0I	EQU	010H	noise
SQSD07	EQU	07					
SQSD08	EQU	08		SQSV00	EQU	00	volume
SQSD09	EQU	09		SQSV01	EQU	01	
SQSD10	EQU	10		SQSV02	EQU	02	
SQSD11	EQU	11		SQSV03	EQU	03	
SQSD12	EQU	12		SQSV04	EQU	04	
SQSD13	EQU	13		SQSV05	EQU	05	
SQSD14	EQU	14		SQSV06	EQU	06	
SQSD15	EQU	15		SQSV07	EQU	07	
SQSTON	EQU	080H	Tones	SQSV08	EQU	08	
				SQSV09	EQU	09	
SQS042	EQU	000H	42 HZ	SQSV10	EQU	10	
SQS083	EQU	004H	83 HZ	SQSV11	EQU	11	
SQS125	EQU	008H	125 HZ	SQSV12	EQU	12	
SQS167	EQU	00CH	167 HZ	SQSV13	EQU	13	
SQS250	EQU	010H	250 HZ	SQSV14	EQU	14	
SQS333	EQU	014H	333 HZ	SQSV15	EQU	15	
SQS500	EQU	018H	500 HZ				
SQS667	EQU	01CH	667 HZ	SQSPAU	EQU	020H	PAUSE
SQS01K	EQU	020H	1 KHZ	SQSCHA	EQU	010H	CHAINING
SQS02K	EQU	024H	2 KHZ	SQSEND	EQU	000H	END OF TABLE
SQSLAA	EQU	0C0H	low impure				
SQSLBB	EQU	0F0H	low impure				
SQSHAA	EQU	0C1H	high impure				
SQSMBB	EQU	0D1H	high impure				

GENERAL EQUATES (1)

macro GMIRAM

internal ram addresses

GZBAND	EQU	020H	flags for controlling processing - vb bands	
GZLIVE	EQU	029H	nr of lives	
GZROUND	EQU	02AH	round nr	
GZMOD3	EQU	02BH	modulo 3 counter etc.	
GZSLCT	EQU	02CH	selected game or game parameters	
GZNSH1	EQU	02DH	name-score	high-score 1
GZNSH3	EQU	02EH		high-score 3
GZNSAR	EQU	02FH		arrow
GZNSN2	EQU	030H		name 2
GZNSN4	EQU	031H		name 4
GZNSN6	EQU	032H		name 6
GZNSC1	EQU	033H		current score 1
GZNSC3	EQU	034H		current score 3
GZNSH2	EQU	035H		high score 2
GZNSH4	EQU	036H		high score 4
GZNSN1	EQU	037H		name 1
GZNSN3	EQU	038H		name 3
GZNSN5	EQU	039H		name 5
GZNSP	EQU	03AH		space
GZNSC2	EQU	03BH		current score 2
GZNSC4	EQU	03CH		current score 4
GZNSL9	EQU	GNSAR		LSS9-values
GZNSNC	EQU	GNSSP		name count
SZOVST	EQU	03DH	overlap status copy	
GZM256	EQU	03EH	modulo 256 counter	
SZMISC	EQU	03FH	miscellaneous flags	

GENERAL EQUATES (2)

macro GME RAM

external ram addresses

SXMISC	EQU	000H
SXMIN5	EQU	001H
SXSECS	EQU	002H
GXINPT	EQU	003H
GXSDPR	EQU	004H
GXSDAD	EQU	005H
GXRND1	EQU	006H
GXRND2	EQU	007H
GXSWAP	EQU	008H
GXEXPY	EQU	009H
GXEXPX	EQU	00AH
GXEXPC	EQU	00BH
GXMJN1	EQU	00CH
GXMJN2	EQU	010H
GXMJN3	EQU	014H
GXMJN4	EQU	018H
GXMAJ1	EQU	01CH
GXMAJ2	EQU	01FH
GXMAJ3	EQU	022H
GXMAJ4	EQU	025H
GXMAJ5	EQU	028H
GXMAJ6	EQU	02BH
GXMAJ7	EQU	02EH
GXMAJ8	EQU	031H
GXMAJ9	EQU	034H
GXMAJA	EQU	037H
GXMAJB	EQU	03AH
GXMAJC	EQU	03DH
GXMJC1	EQU	040H
GXMJC3	EQU	041H
GXMJC5	EQU	042H
GXMJC7	EQU	043H
GXMJC9	EQU	044H
GXMJCB	EQU	045H

GXGM11	EQU	046H
GXGM12	EQU	047H
GXGM13	EQU	048H
GXGM14	EQU	049H
GXGM21	EQU	04AH
GXGM22	EQU	04BH
GXGM23	EQU	04CH
GXGM24	EQU	04DH
GXGM31	EQU	04EH
GXGM32	EQU	04FH
GXGM33	EQU	050H
GXGM34	EQU	051H
GXGM41	EQU	052H
GXGM42	EQU	053H
GXGM43	EQU	054H
GXGM44	EQU	055H
GXGC11	EQU	056H
GXGC13	EQU	057H
GXGC21	EQU	058H
GXGC23	EQU	059H
GXGC31	EQU	05AH
GXGC33	EQU	05BH
GXGC41	EQU	05CH
GXGC43	EQU	05DH
GXGP1Y	EQU	05EH
GXGP1X	EQU	05FH
GXGP2Y	EQU	060H
GXGP2X	EQU	061H
GXGP3Y	EQU	062H
GXGP3X	EQU	063H
GXGP4Y	EQU	064H
GXGP4X	EQU	065H
GXGR1B	EQU	06AH
GXGR1H	EQU	075H
GXGR1V	EQU	075H
GXTEST	EQU	07FH

GENERAL EQUATES (3)

macro GMGEN1 miscellaneous routines

GQINEN	EQU	0FFH	initialization	end of table
GQINII	EQU	0FEH		internal ram itemized
GQINEI	EQU	0FDH		external ram itemized
GQINVI	EQU	0FCH		video ram itemized
GQINIR	EQU	0FBH		internal ram repeat
GQINER	EQU	0FAH		external ram repeat
GQINVR	EQU	0F9H		video ram repeat
GQINCH	EQU	0F8H		chaining in table

GQMIP8	EQU	000H	shadow ram	minor	page 8 table
GQMIP9	EQU	040H		minor	page 9 table
GQMIPA	EQU	080H		minor	page A table
GQMIPB	EQU	0C0H		minor	page B table

GQMIID	EQU	0	shadow ram	minor	transformation = identity
GQMIRX	EQU	1		..	= refl. in x axis
GQMIRY	EQU	2		..	= refl. in y axis
GQMIHT	EQU	3		..	= half turn

GQMI VI	EQU	4	shadow ram	minor	visible
GQMI IV	EQU	0		minor	invisible

GQMAVI	EQU	000H	shadow ram	major	visible
GQMAIV	EQU	080H		major	invisible

Naming Conventions

1. Component Letter

S/T	System (=exec)	bank 0/1	for standard non ROM items use
G/H	General Library	bank 0/1	for general non ROM items use
other	Game-id		

2. Function Letter

Z or I?	Int Ram		
X	Ext Ram		
V	Video Ram (=8244 Ram)		(=Graphics Ram)
D	Rom Table Data		
P/Q	Procedure	bank 0/bank 1	or label
M	Macro		
Q → E	Equate for non-address constant e.g. colour		
S	Section (=segment)		
F	File		

File Name	GFMINS	HFMINS	(MINS=minor)
Section Name	GSMINS	HSMINS	
Entry Name	GPMINS	HPMINS	

Graphics ram minor location	SVMIN2	EQU	4H
Extram shadow minor location	GXMIN2	EQU	010H

Macro Name
Macro File name

Better idea? No component letter
Function letter extension

{XGM'L
{XGM'C
{XGM'Y

procedure QSRMIN
QSRMAJ

addresses

{VMI P
{VMI Y {XM JY
XMI Y {XM JC

OR procedures P1
P1

equates

ECOLR EMJRED
E'CTRL EFPS

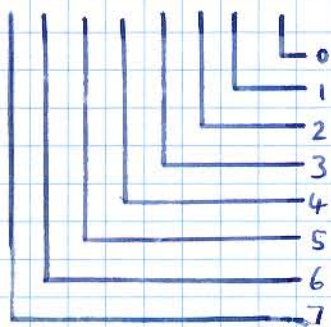
SUMMARY OF MACROS

S1	SMKEYS	SQ..	keycodes
S2	SM COLS	SQ MI/MA/BG/GR	colours
S3	SM VRAM	SV....	video ram addresses
S4	SMVREG	SQV...	video ram register control (except sound)
S5	SMEXEC	various	executive routines + sounds
S6	SMPORT	SQ	port settings
S7	SM SOUN	SQS...	sound control
S8	SM		

G1	GMIRAM	GZ....	internal ram
G2	GMERAM	GX....	external ram
G3	GMGEN1	GQ IN..	general routines : - initialization tables
	+	GQ MI/MA ..	- shadow ram control
	+	GQ	-
	GM		

8048

7 6 5 4 3 2 1 0



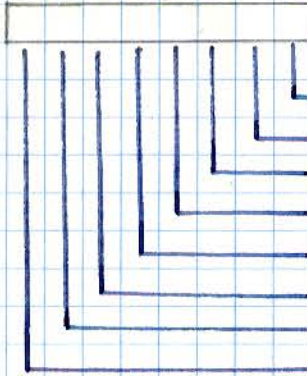
PSW

program status word

1	C	carry
2	AC	auxiliary carry
4	F0	Flag 0
8	BS	register bank select
10	1	
20	SP2	} stack cell pointer
40	SP1	
80	SP0	

PORT ASSIGNMENTS

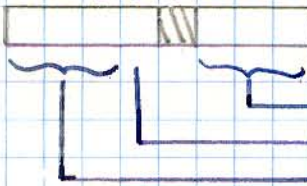
7 6 5 4 3 2 1 0



PORT 1

game rom select 00 = 6K-8K 02 = 2K-4K
 01 = 4K-6K 03 = 0K-2K
 keyboard+hand control enable 0 = enable 1 = disable
 select video (8244) ram 0 = active
 select external ram 0 = active
 hand control input enable 0 = active
 one way block transfer enable 1 = active
 force video high 0 = active

7 6 5 4 3 2 1 0

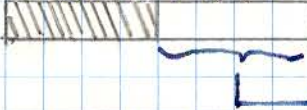


PORT 2

with keyboard enabled

keyboard row select 1 = active
 valid key data 0 = active
 keyboard column data 0 = active

7 6 5 4 3 2 1 0

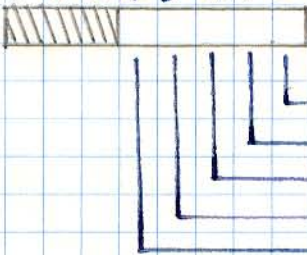


PORT 2

hand control select

hand control nr. select 0-7

7 6 5 4 3 2 1 0



BUS

up 0 = active
 right 0 = active
 down 0 = active
 left 0 = active
 action button 0 = active

PORT1 B7, B6, B5, B4 = select 8244 ram

E7, E6, E5, E4 = select one way transfer

AF, AE, AD, AC = select EXTRAM

INTERNAL RAM

00	RBO:R0
01	RBO:R1
02	RBO:R2
03	RBO:R3
04	RBO:R4
05	RBO:R5
06	RBO:R6
07	RBO:R7
08	STACK CELL #0
09	
0A	STACK CELL #1
0B	
0C	STACK CELL #2
0D	
0E	STACK CELL #3
0F	
10	STACK CELL #4
11	
12	STACK CELL #5
13	
14	STACK CELL #6
15	
16	STACK CELL #7
17	
18	RBI:R0
19	RBI:R1
1A	RBI:R2
1B	RBI:R3
1C	RBI:R4
1D	RBI:R5
1E	RBI:R6
1F	RBI:R7

IBAND

GZBAND

ILIVE

GZLIVE

IROWN

GZROWN

IMOD3

GZMOD3

ISLCT

GZSLCT

INSHI

GZNSHI

INSH3

GZNSH3

INSAR

GZNSAR=L9

INSN2

GZNSN2

INSN4

GZNSN4

INSN6

GZNSN6

INSCI

GZNSCI

INSC3

GZNSC3

INSH2

GZNSH2

INSH4

GZNSH4

INSN1

GZNSN1

INSN3

GZNSN3

INSN5

GZNSN5

INSSP

GZNSSP=NC

INSC2

GZNSC2

INSC4

GZNSC4

IOVST

SZOVST

IM256

GZM256

IMISC

SZMISC

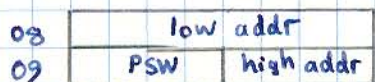
20	flags for processing/vb bands	
21		
22		
23		
24		
25		
26		
27		
28		
29	nr of lives	
2A	round	
2B	modulo 3 extended	
2C	selected game	
2D	name-score	high 1
2E		high 3
2F		arrow
30		name 2
31		name 4
32		name 6
33		curr. 1
34		curr. 3
35		high 2
36		high 4
37		name 1
38		name 3
39		name 5
3A		space
3B		curr. 2
3C		curr. 4
3D	overlap status	
3E	modulo 256	
3F	miscellaneous	

SZOVST = OVSTAT

GZM256 = MODU60

SZMISC = MISC

INTERNAL RAM



STACK CELL #0 = 08+09



GZMOD3 = 2B

modulo 3 etc.

mod 3 value 0, 1, 2, 0, 1, 2...
0
quotient 3 value 0, 0, 0, 1, 1, 1, 0, 0, ...
mod 5 value
quotient 5 value



GZSLCT = 2C

game select

key selected
location may be reused with bit flags
nr of players 0, 1, 2 (0 = demo)
difficulty player #1
difficulty player #2
variation nr

INTERNAL RAM

name-score structure

H1	H2	H3	H4	AR	N1	N2	N3	N4	N5	N6	SP	C1	C2	C3	C4	name
2D	35	2E	36	2F	37	30	38	31	39	32	3A	33	3B	34	3C	address

6	7	8	9	→	N	A	M	E	?	?		0	1	2	3
---	---	---	---	---	---	---	---	---	---	---	--	---	---	---	---

1.1	3.1	1.2	3.2	1.3	3.3	1.4	3.4	2.1	4.1	2.2	4.2	2.3	4.3	2.4	4.4	group maj
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----------

1	1	1	1	1	var.	var.	var.	var.	var.	var.	1	1	1	1	1	lss9
---	---	---	---	---	------	------	------	------	------	------	---	---	---	---	---	------

grp	name	addr	lss9
-----	------	------	------

1.1	H1	2D	1
1.2	H3	2E	1
1.3	AR=L9	2F	1
1.4	N2	30	var
2.1	N4	31	var
2.2	N6	32	var
2.3	C1	33	1
2.4	C3	34	1
3.1	H2	35	1
3.2	H4	36	1
3.3	N1	37	var
3.4	N3	38	var
4.1	N5	39	var
4.2	SP=NC	3A	1
4.3	C2	3B	1
4.4	C4	3C	1

L9=lss9 settings, for variable lss9, starting in bit 7

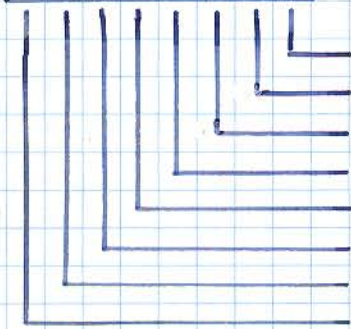
NC= number of characters input

Initial settings for line 200

1.1	H1	9CH	0
1.2	H3	9CH	0
1.3	L9	00H	no lss9=1
1.4	N2	04H	?
2.1	N4	04H	?
2.2	N6	04H	?
2.3	C1	9CH	0
2.4	C3	9CH	0
3.1	H2	9CH	0
3.2	H4	9CH	0
3.3	N1	04H	?
3.4	N3	04H	?
4.1	N5	04H	?
4.2	NC	00H	no characters input
4.3	C2	9CH	0
4.4	C4	9CH	0

INTERNAL RAM

7 6 5 4 3 2 1 0



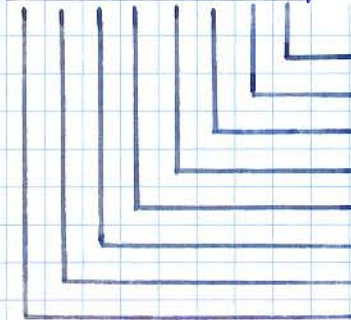
bit	hex value
0	1
1	2
2	4
3	8
4	10
5	20
6	40
7	80

SZMISC = 3F

miscellaneous

Interrupt proc: 0=exec, 1= local to games prog
TV system 0=PAL, 1=NTSC
RAM sound
ROM page 5 sound
ROM page 6 sound
EXEC sound
Sound Processing by executive
Data Packets processing by executive

7 6 5 4 3 2 1 0



0	1
1	2
2	4
3	8
4	10
5	20
6	40
7	80

SZOVST = 3D

overlap status (copy of GVOVST)

minor #0
minor #1
minor #2
minor #3
grid vertical
grid horizontal
(chip expander)
major system

EXTERNAL RAM

XMJY1

SXMISC	00	
SXMIN5	01	timer minutes
SXSECS	02	timer seconds
GXINPT	03	input data
GXSDPR	04	sound priority
GXSDAD	05	sound address
GXRND1	06	random term series
GXRND2	07	random term series
GXSWAP	08	minor-major sharing
GXEXPY	09	explosion y
GXEXPX	0A	explosion x
GXEXPC	0B	explosion counter
GXMIN1	0C	
	0D	MINOR #1
	0E	
	0F	
GXMIN2	10	
	11	MINOR #2
	12	
	13	
GXMIN3	14	
	15	MINOR #3
	16	
	17	
GXMIN4	18	
	19	MINOR #4
	1A	
	1B	

SXMISC = MISC2
SXMIN5 = MINTES
SXSECS = SECNDS

GXMAJ1	1C	
	1D	MAJOR #1
	1E	
GXMAJ2	1F	
	20	MAJOR #2
	21	
GXMAJ3	22	
	23	MAJOR #3
	24	
GXMAJ4	25	
	26	MAJOR #4
	27	
GXMAJ5	28	
	29	MAJOR #5
	2A	
GXMAJ6	2B	
	2C	MAJOR #6
	2D	
GXMAJ7	2E	
	2F	MAJOR #7
	30	
GXMAJ8	31	
	32	MAJOR #8
	33	
GXMAJ9	34	
	35	MAJOR #9
	36	
GXMAJA	37	
	38	MAJOR #10
	39	
GXMAJB	3A	
	3B	MAJOR #11
	3C	
GXMAJC	3D	
	3E	MAJOR #12
	3F	
GXMJC1	40	MAJOR
GXMJC3	41	COLOURS
GXMJC5	42	AND
GXMJC7	43	LSS
GXMJC9	44	INCREMENT
GXMJCB	45	
GXGM11	46	
GXGM12	47	GROUP #1
GXGM13	48	LSS'S
GXGM14	49	
GXGM21	4A	
GXGM22	4B	GROUP #2
GXGM23	4C	LSS'S
GXGM24	4D	

XMJCI

XGM11

GXGM31	4E	
GXGM32	4F	GROUP #3
GXGM33	50	LSS'S
GXGM34	51	
GXGM41	52	
GXGM42	53	GROUP #4
GXGM43	54	LSS'S
GXGM44	55	
GXGC11	56	→ XGC11
GXGC13	57	GROUP
GXGC21	58	MAJOR
GXGC23	59	COLOURS
GXGC31	5A	AND
GXGC33	5B	LSS'S
GXGC41	5C	
GXGC43	5D	
GXGP1Y	5E	GROUP #1
GXGP1X	5F	→ XGMY1
GXGP2Y	60	GROUP #2
GXGP2X	61	
GXGP3Y	62	GROUP #3
GXGP3X	63	
GXGP4Y	64	GROUP #4
GXGP4X	65	
	66	
	67	
	68	
	69	
GXGRIB	6A	
	6B	GRID BOTTOM ROW
GXGRIH	6C	
	6D	GRID
	6E	HORIZONTAL
	6F	
	70	
	71	
	72	
	73	
	74	
GXGRIV	75	
	76	GRID
	77	VERTICAL
	78	
	79	
	7A	
	7B	
	7C	
	7D	
	7E	
GXTEST	7F	test data location

EXTERNAL RAM

7	6	5	4	3	2	1	0
Y	Y	Y	Y	Y	Y	Y	Y
X	X	X	X	X	X	X	X
PAGE	B	G	R	D	S	X	9
ADDRESS				V	TRAN		

I	Y	Y	Y	Y	Y	Y	Y
X	X	X	X	X	X	X	X
C2	C3	K	K	K	K	K	K

B	G	R	C1	↑
---	---	---	----	---

next major colour etc

L1	L2	L3	L4	L5	L6	L7	L8
----	----	----	----	----	----	----	----

B	G	R	L9	↑
---	---	---	----	---

next group major colour etc

O	Y	Y	Y	Y	Y	Y	Y
X	X	X	X	X	X	X	X

GXMINI = 0CH minor

see general equates for interpretation
n.b. X coord is line coordinate

GXMAJI = 1CH major

y coord I = invisibility bit, not applicable to group major
X coord PIXEL COORDINATE = 2 × line coordinate
K = key code

GXMJCI = 40H major colour

C1, C2, C3 = LSS increment

C1 = 1 pixel
C2 = 2 pixel
C3 = 4 pixel } max 5 pixels
B, G, R = blue, green, red

GXGMII = 46H group major
LSS value

GXGCII = 56H group major colour
B, G, R = blue, green, red. L9 = LSS9 value

GXGPIY = 5E group major coords, whole group
y coord
X coord PIXEL COORDINATE = 2 × line coordinate

VIDEO RAM

SVMINI	00	MINOR #1	SVGP11	40	GRP 1.1	SVMIP1	80	MINOR #1	SVGR11	C0
	02			42			82			C2
SVMIN2	04	MINOR #2	SVGP12	44	GRP 1.2		84	PATTERN		C4
	06			46			86			C6
SVMIN3	08	MINOR #3	SVGP13	48	GRP 1.3	SVMIP2	88	MINOR #2		C8
	0A			4A			8A			CA
SVMIN4	0C	MINOR #4	SVGP14	4C	GRP 1.4		8C	PATTERN		CC
	0E			4E			8E			CE
SVMAJ1	10	MAJOR #1	SVGP21	50	GRP 2.1	SVMIP3	90	MINOR #3	SVGR13	D0
	12			52			92			D2
SVMAJ2	14	MAJOR #2	SVGP22	54	GRP 2.2		94	PATTERN		D4
	16			56			96			D6
SVMAJ3	18	MAJOR #3	SVGP23	58	GRP 2.3	SVMIP4	98	MINOR #4		D8
	1A			5A			9A			DA
SVMAJ4	1C	MAJOR #4	SVGP24	5C	GRP 2.4		9C	PATTERN		DC
	1E			5E			9E			DE
SVMAJ5	20	MAJOR #5	SVGP31	60	GRP 3.1		A0	REGISTERS (see below)	SVGR15	E0
	22			62			A2		E2	
SVMAJ6	24	MAJOR #6	SVGP32	64	GRP 3.2		A4		E4	
	26			66			A6		E6	
SVMAJ7	28	MAJOR #7	SVGP33	68	GRP 3.3		A8		E8	
	2A			6A			AA		EA	
SVMAJ8	2C	MAJOR #8	SVGP34	6C	GRP 3.4		AC		EC	
	2E			6E			AE		EE	
SVMAJ9	30	MAJOR #9	SVGP41	70	GRP 4.1		B0		F0	
	32			72			B2		F2	
SVMAJA	34	MAJOR #10	SVGP42	74	GRP 4.2		B4		F4	
	36			76			B6		F6	
SVMAJB	38	MAJOR #11	SVGP43	78	GRP 4.3		B8		F8	
	3A			7A			BA		FA	
SVMAJC	3C	MAJOR #12	SVGP44	7C	GRP 4.4		BC		FC	
	3E			7E			BE		FE	

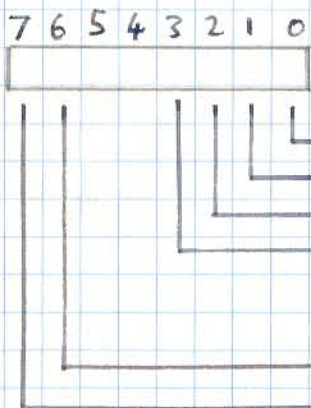
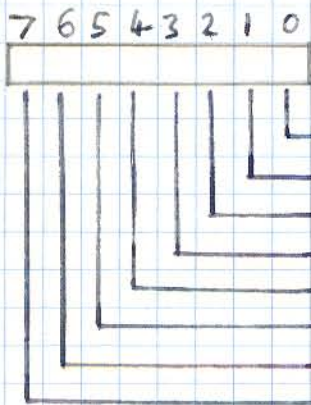
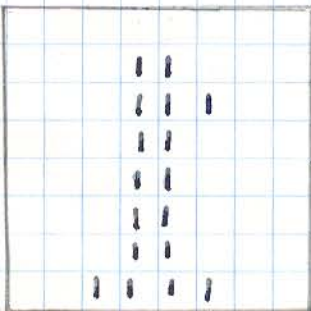
VMIYI
VMJIYI
VGMII
VMIPI

SVCTRL	A0	control	R-W
SVSTAT	A1	control status	R-only
SVOVST	A2	overlay	R-W
SVCOLR	A3	colour	W-only
SVYSTR	A4	y-strobe	R-only
SVXSTR	A5	x-strobe	R-only
	A6		not used
SVSDSR	A7	sound	W-only
	A8	shift	W-only
	A9	register	W-only
SVSDCT	AA	sound control	R-W

VIDEO RAM

Y	Y	Y	Y	Y	Y	Y	Y
X	X	X	X	X	X	X	X
		B	G	R	D	S	X9

Y	Y	Y	Y	Y	Y	Y	Y
X	X	X	X	X	X	X	0
L	L	L	L	L	L	L	L
		B	G	R	D	S	X9



SVMIN1 minor

y coord
x coord (line coord)
blue, green, red, double size, smoothing, X9

SVMAJ1 major (also group major)

y coord
x coord (line coord)

SVMIPI minor pattern

reflected bit pattern of object required.

SVCTRL = 0A0H

ehi	enable horizontal interrupt
fps	forced position strobe
esi	enable sound interrupt
egr	enable grid
eeo	enable external overlap
edp	enable display
den	dot enable
grw	grid wide

SVSTAT = 0A1H

hst	horizontal status
pss	position strobe status
sns	sound needs service
vst	vertical status
.	
exo	external overlap
mjo	major overlap

VIDEO RAM

Grid element addressing

→ increasing byte locations horiz, vertical and bottom

GRIV CO/0	C1/0	C2/0	C3/0	C4/0	C5/0	C6/0	C7/0	C8/0
E0/0	E1/0	E2/0	E3/0	E4/0	E5/0	E6/0	E7/0	E8/0
CO/1	C1/1	C2/1	C3/1	C4/1	C5/1	C6/1	C7/1	C8/1
E0/1	E1/1	E2/1	E3/1	E4/1	E5/1	E6/1	E7/1	E8/1
CO/2	C1/2	C2/2	C3/2	C4/2	C5/2	C6/2	C7/2	C8/2
E0/2	E1/2	E2/2	E3/2	E4/2	E5/2	E6/2	E7/2	E8/2
CO/3	C1/3	C2/3	C3/3	C4/3	C5/3	C6/3	C7/3	C8/3
E0/3	E1/3	E2/3	E3/3	E4/3	E5/3	E6/3	E7/3	E8/3
CO/4	C1/4	C2/4	C3/4	C4/4	C5/4	C6/4	C7/4	C8/4
E0/4	E1/4	E2/4	E3/4	E4/4	E5/4	E6/4	E7/4	E8/4
CO/5	C1/5	C2/5	C3/5	C4/5	C5/5	C6/5	C7/5	C8/5
E0/5	E1/5	E2/5	E3/5	E4/5	E5/5	E6/5	E7/5	E8/5
CO/6	C1/6	C2/6	C3/6	C4/6	C5/6	C6/6	C7/6	C8/6
E0/6	E1/6	E2/6	E3/6	E4/6	E5/6	E6/6	E7/6	E8/6
CO/7	C1/7	C2/7	C3/7	C4/7	C5/7	C6/7	C7/7	C8/7
E0/7	E1/7	E2/7	E3/7	E4/7	E5/7	E6/7	E7/7	E8/7
GRIB DO/0	D1/0	D2/0	D3/0	D4/0	D5/0	D6/0	D7/0	D8/0