Software compatibility between Oric and Apple 2

An article by Ventzislav T. *and reactions compiled from newsgroup <comp.sys.oric> by André C.*

First of all - there is no compatibility at software level between the Oric and the Apple 2 series of microcomputers, but both computers share many common specifics. Compatibility is possible between computers with similar aspects (processor, memory management, input-output devices etc.). In Apple 2 and Oric there are enough common parts giving a chance some software from the one to be converted to the other and vice versa. At Basic and Dos levels this is easy enough for the causal Oric/Apple 2 user to be able to convert programs and scripts between Apple 2 and Oric, so I won't discuss them here. The assembly level is the most problematic, since the Roms of both computers are written in assembler and these machine language programs bang the hardware directly. Here I will mark the basis, without pretending, the given info is fully accurate and/or descriptive, but it should be a good start for developing on the idea.

Before describing the question about the software compatibility, I have to note that some differences in the organization of the two computers sometimes makes the conversion of programs from the one computer to the other almost impossible. The differences are biggest in the controls of the graphics and sound. In the Oric, there is a special co-processor for sound - AY-3-8912, while in the Apple 2 there is a trigger, with which the membrane of the speaker is moved from one to other of the two possible positions. In the Oric, the control of the memory and the graphics is put to the ULA (Universal Logic Array) chip, which on the Apple 2 is different. On the Apple 2 there are 4 analog inputs, 4 digital inputs and outputs, while on the Oric they have to be simulated by software with the (VIA 6522). Apple 2 programs, which use the double high resolution, and these, with requirements of extended memory are extremely hard for conversion, due to the different bank switching/code jumping technologies used.

With help from the tables given below, it is possible to convert only some assembly language programs. If the program uses the graphical memory or sound effects, the subroutines must be rewritten, especially for the given case. Since the Oric does not have built-in monitor, programs on the Apple 2, which use monitor subroutines, have to be rewritten too. With programs controlling the disk drive on Oric the interrupts must be disabled (SEI), because the information written or read can get inaccurate. Another problem is that the Apple 2 does not have interrupts, so programs using interrupts must have to be totally rewritten. On the Apple 2 the control of the different modes is done by software switches, which are missing on the Oric. Simulation of these switches has to be done with jumping to subroutines of the Basic interpreter, which do almost the same. Before trying to rewrite a program from one to the other computer you must be sure that you know the organization of the both computers equally well, and you can easily move from one to the other. Despite all this, it is more reasonable for a start to be done conversion of system tools and applications, while games and demos are much harder (But not impossible).

In the end I will notice that there is more point in converting and translating programs from the Apple 2 to the Oric, since there is much less software for it.

ZERO PAGE

Apple	Oric
$0A-$0C	$21-$23
$200-$2FF	$35-$85
$67-$68	$9A-$9B
$69-$6A	$9C-$9D
$6B-$6C	$9E-$9F
$6D-$6E	$A0-$A1
$6F-$70	$A2-$A3
$73-$74	$A6-$A7
$75-$76	$A8-$A9
$77-$7A	$AA-$AD
$3F5-$3F7	$2FC-$2FD
$E4	$213
$E0-$E1	$219
$E2	$21A
$36-$37	$238-$23A
$38-$39	$23B-$23D
$21	$257
$24	$268
$25	$269
$22	$27E
$28-$29	$19-$13

Oric	-	BASIC Address
CALL	-	$E946
CHAR	-	$F12D
CIRCLE	-	$F37F
CLEAR	-	$C70D
CLOAD	-	$E85B
CONT	-	$C9AD
CLS	-	$CCCE
CSAVE	-	$E909
CURMOV	-	$F0FD
CURSET	-	$F0C8
DATA	-	$CA3C
DEF FN	-	$D4BA
DIM	-	$D17E
DOKE	-	$D967
DRAW	-	$F110
EDIT	-	$C692
END	-	$C973
EXPLODE	-	$FACB
FILL	-	$F2C8
FOR	-	$C855
GET	-	$CD46
GOSUB	-	$C9C8
GOTO	-	$C9E5
GRAB	-	$E8E7
HIMEM	-	$EBCE

HIRES	-	$EC33
IF	-	$CA70
INK	-	$F21D
INPUT	-	$CD55
LET	-	$CB1C
LIST	-	$C748
LLIST	-	$C7FD
LORES	-	$D9DE
LPRINT	-	$C809
MUSIC	-	$FC18
NEW	-	$C6EE
ON	-	$CAC2
PAPER	-	$F204
PATTERN	-	$F11D
PING	-	$FA9F
PLAY	-	$FBD0
PLOT	-	$DA51
POINT	-	$F1C8
POKE	-	$D94F
POP	-	$CA12
PRINT	-	$CBAB
PULL	-	$DAA1
READ	-	$CD89
RECALL	-	$E9D1
RELEASE	-	$EC0C
REM	-	$CA99
RESTORE	-	$C952
RETURN	-	$CA12
RUN	-	$C9BD
SHOOT	-	$FAB5
SOUND	-	$FD40
STOP	-	$C971
STORE	-	$E987
TEXT	-	$EC21
TROFF	-	$CD10
TRON	-	$CD16
UNTIL	-	$DAA1
WAIT	-	$D958
ZAP	-	$FAE1
!	-	$CD13
&	-	$DADB

Apple 2 (Disk controller)		**Oric (Disk controller)**
$C080	-	$310
$C081	-	$311
$C082	-	$312
$C083	-	$313
$C084	-	$314
$C085	-	$315
$C086	-	$316
$C087	-	$317
$C088	-	$318
$C089	-	$319
$C08A	-	$31A
$C08B	-	$31B
$C08C	-	$31C
$C08D	-	$31D
$C08E	-	$31E
$C08F	-	$31F

Apple 2(monitor)	**Oric**
$ED24	$E0C5
$FD0C	$C5E8
$FD6F	$C592
$FDED	$CC9D
$FE89/$FE93	$E93D
$FB2F	$F9C9
$C000	$2DF
$C010	$2DF
$C083	$381 (With 16 KB RAM expansion)

Graphic memory:

Apple 2	**Oric**
TEXT/LOWRES 1 $0400-$07FF	$BB80-$BFE0
TEXT/LOWRES 2 $0800-$0BFF	
HIRES 1 $2000-$3FFF	$A000-$BFE0
HIRES 2 $4000-$5FFF	

Double text/lowres and double highres modes on the Apple use twice amount of memory with half of that in the standard Ram, and the other half in the extended memory, accessed with software switches - $C000, $C001, $C018. But I won't describe them here.

Note that the Apple uses different memory locations for the TEXT/LOWRES modes, allowing easy switch between text/graphic modes, while on the Oric, the graphic and text modes overlap each other, and if you need to save the text while switching to HIRES, you need to copy the screen data, and later sending it back, after switching again to TEXT, if there is a demand for save of the text on the screen. Also the Apple 2 have two pages for each mode, which are easily switched between each other ($C051 for first page, $C052 for second page), with just one read/write to the software switches, making it easy for double buffered animation, while the Oric needs copying of data to the graphic memory, thus slow. Organization of the graphic memory is different too. On the Apple 2, the first row of text/graphic is on address $0400-$0427/$4000-$4027, but the second is on $0480-$04A7/$4080-$40A7, the third $0500-$527/$4100-$4127 etc. i.e. non-linear, while on the Oric the first row is $BB80/$A000, the second is on $BBA8/$A0A8 etc. i.e. linear.

Producing sound on the Apple 2 is done by accessing the $C030 memory cell, which is software switch, forcing the sound membrane to move. Reading/writing this switch with different frequency produces sounds with different pitch, and this is all you need to know about the Apple 2 sound capabilities. Needless to say - this takes all the CPU resources, while on the Oric you just can program the AY-3-8912 for the job, but you are not able to do it directly, which makes sound converting little more complex. The easier way is to use the PLAY subroutine from the Rom.

That's about the theoretical part. Now some explanations how to use this in practise. The hardest way is to directly disassemble the program on your Oric/Apple 2 and then start rewriting it on the opposite platform, taking in account whenever the code use hardware/ROM specific routines, rewriting them for the target. Problems will arise when you replace routines containing access to software switches, which will obviously be more/less bytes. If it is less that's easy - just fill the unused bytes with NOPs, but if it is more, that will make non-relocatable code really hard to translate and the mess will raise fast.

More easier is to write a library (or second Rom), which contains the most used subroutines of the source platform rewritten for the target platform and a program which checks if the translated code makes jump to these

routines and if so, to replace this with jump to the library (This program can be written on any computer platform). This way you replace 2 bytes with 2 bytes. Second - check if the program uses the software switches, and replace these routines with your own routines, which can be in the library too, which is the better choice and keeps the code size too. Third - check if the program attempts to self modify/makes direct jumps in own code or some other relocatable unfriendly code and replace this code with indirect jumps, non self modifying code etc.., so it will work on the target platform from different address. All this can be done by a program (Not by hand).

Another bet is writing a software simulator, which starts the program in it's native unchanged form, from the source platform, on the target platform. While executing a program, the simulator checks if there is a hardware specific code and replaces it with own code and then jumps to the next opcode, this done by interrupt routines. With this, you can execute Apple 2 programs directly on the Oric, without modifying them, thus not violating their copyright.

For native Basic programs - interpreter of Applesoft Basic on the Oric and Oric Basic interpreter on the Apple 2 is a reasonable choice too.

For an end, I will say that I really would love to have SWEET 16 on the Oric. 2003 © by Ventzislav T. Any comments/corrections/additions/flames etc.. will be appreciated. Thanks in advance.

Fabrice F.: Thanks a lot, Ventzislav, that is definitely useful. I'm sure Sweet16 wouldn't be too hard to port on the Oric... If I help you porting it, would you help me port Sargon III and Robot Odissey ?-)

Ventzislav T.: [Sweet16] As I stated in the article, for a start it's better to begin with applications software, because they mostly use the processor, rather than fancy graphic and sound effects. The Sweet16 source code was available on the Merlin assembler disk, but I do not have it anymore. With having the original source core, the port is even easier. [port Sargon III and Robot Odissey] I am doing a research on the most used sound and graphic routines on the Apple 2 and about their equivalents on the Oric. With the sound, there are not as many problems as with the graphic. The Apple 2 monochrome highres mode is 280/192 pixels, while on the Oric the graphic is 240*200 (no difference between monochrome and colour), but if you interpret the Apple 2 display as colour, the resolution is 140/192, which is still the same even in double highres, but there are 4bits per colour pixel (16 colours total) in the double highres. So far, the Apple 2 displays 7 bits/pixels per byte (40*7=280), while the Oric only 6 bits/pixel per byte (40*6=240). Two possible compromises are interpreting the Apple 2 graphics as colour and try to implement them as colour on the Oric too, or omitting 1 pixel per byte on the Oric graphics display and not using any colour, with directly writing the graphic bytes to the video memory. The first is preferable, because then you can have the whole picture, without crushing the display, but it is harder to implement too. The LOWRES modes (GR and GR2 on the Apple 2) are easy to simulate by using the mosaic characters present in the Oric. Later I will do another comparison table for Apple 2 characters in the lowres and Oric mosaic equivalents.

Kamelito: What we need first IMHO, is a good 6502 interactive decompiler for the Apple2. Once you have the Apple2 binary, the decompiler (sourcer, disassembler...) will produce an ASM file and it would be then easier to port an Apple2 software on the Oric. The decompiler should replace access to the Rom or Apple Hardware with label for a better understanding of the source. If in plus it could recognize graphics, etc that would be perfect, like the Atari interactive disassembler <http://ourworld.compuserve.com/homepages/ebacher/>. This disassembler could also recognize Oric specificity to port Oric software to the Apple2. I look on the Internet and was not able to find such a program. .../... Oops, I just noticed will not trying it that the Atari interactive disassembler also support the Oric, any chance to add the Apple2?

Fabrice F.: [Sweet16 source code] Perhaps Carsten has this source code? I think he said there was an article about Sweet16, written by Steve himself... .../... Well, now I remember Carsten indeed posted the disassembly of the 300 bytes of Sweet16... <http://www.gno.org/pub/apple2/doc/hardware/motherboards/sweet16.txt>.

The only problem is that most of the 32 first bytes of page 0 are already used on the Oric... However, there is a contiguous area for 32 zeropage-bytes : in the entry buffer ($35-$84), so it should be rather easy to port it... For example, just say that:

```
R0L     EQU  $60
R0H     EQU  $61
R14H    EQU  $7D
R15L    EQU  $7E
R15H    EQU  $7F
```

Now you just need a SAVE and a RESTORE routine, but if you want the best compatibility, you just need to change the ones found in the Apple2 Rom with Oric-compatible zeropage locations:

```
SAVE:    sta $80
         stx $81
         sty $82
         php
         pla
         sta $83
         tsx
         stx $84
         cld
         rts
RESTORE: lda $83
         pha
         lda $80
         ldx $81
         ldy $82
         plp
         rts  ; well, S is not restored in this Apple2 routine.
```

That is it, Sweet16 on the Oric... Do you have any Sweet16 application to run? Cheers, Fabrice