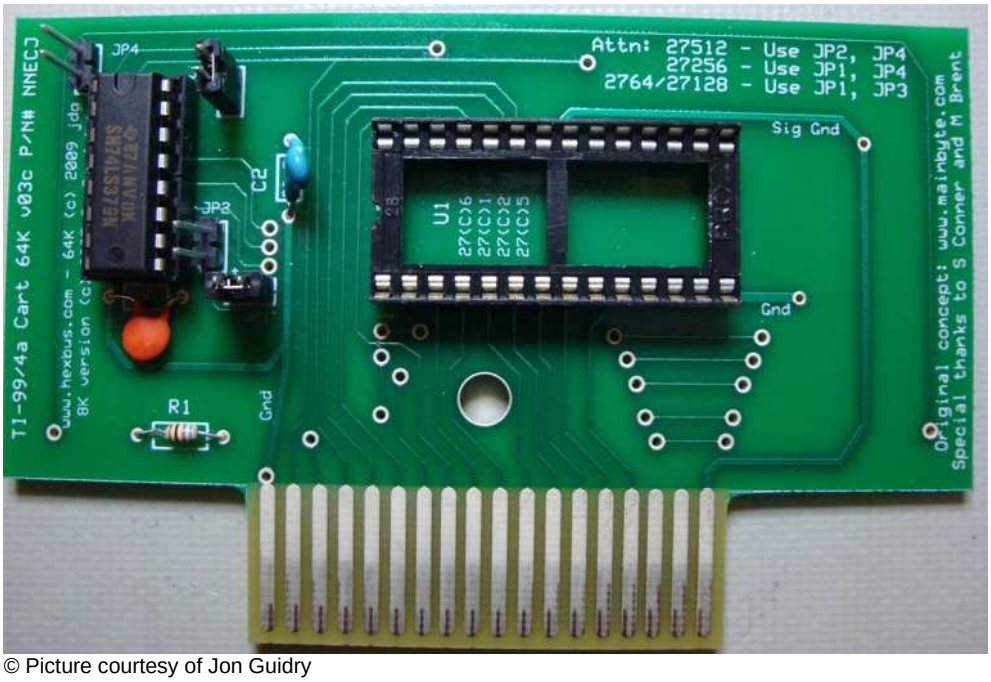
Tutorial: building a multi-bank ROM image

Introduction

This tutorial describes the steps involved for building a multi-bank cartridge ROM image for the Texas Instruments TI-99/4A Home Computer.

As an example we'll build a Pitfall! cartridge ROM image; starting from the assembly source. When done, we will have a 32K ROM image containing four 8K (>6000->7FFF) banks. Burned on EPROM it can be used with the new 16K/32K/64K cartridge PCBs[1] recently made available by Jon Guidry.

Our target platform: the 16K/32K/64K cartridge PCB



© Picture courtesy of Jon Guidry

Note that all steps described in the tutorial will be done on a windows based cross-development system. The main reason is, that the cross-assembler I use is only available for Windows.

---

[1] http://www.hexbus.com

Requirements

You will need a recent version of the windows based cross–assembler **winasm99** which is delivered as part of the win994A emulator package[2]. I used winasm99 v3.8

Of course you'll also need the Pitfall! source code[3] and my deref command-line utility[4]. The latter is mainly used for handling code-realignment issues. More on that later.

---

[2] http://www.99er.net/win994a.shtml

[3] http://www.atariage.com/forums/topic/158467-pitfall-binaries-and-source-code/

[4] http://www.atariage.com/forums/topic/153704-ti-994a-development-resources/

A few hints

Pre-processor directives

I found out that winasm99 supports pre-processor directives, a feature which I believe is not officially documented (nor supported?). Basically I created a main file for each bank and added the equates below. Depending on the bank I then set the corresponding equate to 1.

```
*************************************************************
* Some build directives for building the 2nd bank (BANK1)
********@*****@*********************@**************************
BANK0   EQU   0
BANK1   EQU   1
BANK2   EQU   0
BANK3   EQU   0
```

In the next example the source code between 'IF BANK1' and 'ENDIF' is only assembled if the BANK1 equate equals 1. So I can include (COPY) the same source file in multiple projects/main files and control how it gets assembled by setting the corresponding BANK equates to 1 or 0.

```
DRAWBJ  LI    R1,OVLAY4
        BL    @SPRITE               ; Put overlay sprite "yellow ground" on screen
        BL    @GVRAM
        DATA  VRETRN,RETADR,2       ; Restore Return address ...
        MOV   @RETADR,R0
*--------------------------------------------------------------
   IF BANK1
        CI    R0,DRAWG1             ; Called from DRAWG ?
        JEQ   DRAWBZ                ; Yes, then Return to DRAWG1
        LI    R1,>6002              ; Select BANK2
        LI    R2,PFRETN             ; Jump-back to PFALL in BANK2
        B     @GOBANK               ; Switch bank 2
   ENDIF
*--------------------------------------------------------------
DRAWBZ  B     *R0                   ; ... and Return
```

Bank-switch trampoline code

What you see in the example, is a bank-switch from BANK 1 (2nd bank) to BANK 2 (3rd bank). This is accomplished by calling the GOBANK subroutine which in turn writes the bank-switch code to scratch-pad memory and executes it.

```
*--------------------------------------------------------------
* ROM BANK SELECTION ROUTINE (inversed method for 379 carts)
*--------------------------------------------------------------
* R1=>6006,>6004,>6002,>6000          ; BANK0, BANK1, BANK2, BANK3
* R2=Address in bank
*--------------------------------------------------------------
SWBANK  EQU   >8300                   ; Somewhere in scratchpad RAM
GOBANK  LI    R0,>04E0                ; CLR ...
        MOV   R0,@SWBANK
        MOV   R1,@SWBANK+2            ; ... CLR R1 -> Select BANK0 ... BANK3
        LI    R0,>0460                ; B   ...
        MOV   R0,@SWBANK+4
        MOV   R2,@SWBANK+6            ; ... B (adddress in R2)
        B     @SWBANK                 ; Now run generated code
```

BANK 0 >6000->7FFF	BANK 1 >6000->7FFF	BANK 2 >6000->7FFF	BANK 3 >6000->7FFF
Cartridge Header	**Cartridge Header**	**Cartridge Header**	**Cartridge Header**
write >6006	write >6004	write >6002	write >6000
ROMC.bin ROMC.lst	ROMD.bin ROMD.lst	ROME.bin ROME.lst	ROMF.bin ROMF.lst

Cartridge header

Note that the cartridge header must be present in each bank, as the TI-99/4A can fire up a random bank on reset. In each bank the cartridge header points to a routine that switches to BANK 0 and triggers the initialisation routine. Below is a more complete example, it's part of the main file for BANK 1 (2nd bank):

```
        AORG  >6000
SFIRST  EQU   $
SLOAD   EQU   $
*****************************************************************
* Some build directives
********@*****@*********************@**************************
BANK0   EQU   0
BANK1   EQU   1
BANK2   EQU   0
BANK3   EQU   0
*****************************************************************
* Some equates with addresses of objects in other banks
********@*****@*********************@**************************
        COPY  "D:\Projekte\pitfall\tms9900\pitfall_bank_equates.a99"
*****************************************************************
* Cartridge header
********@*****@*********************@**************************
GRMHDR  BYTE  >AA,1,1,0,0,0
        DATA  PROG
        BYTE  0,0,0,0,0,0,0,0
PROG    DATA  0
        DATA  KICKBK
        BYTE  8
        TEXT  'PITFALL!'
*****************************************************************
*  PITFALL BANK 1
********@*****@*********************@**************************
        COPY  "D:\Projekte\pitfall\tms9900\pitfall.a99"
*****************************************************************
* Kickstart BANK0
********@*****@*********************@**************************
KICKBK  LWPI  WSSPC1                ; Load main workspace
        LIMI  0
        LI    R1,>6006              ; Select BANK0
        LI    R2,BOMAIN             ; B @MAIN (skip cartridge header)
        B     @GOBANK               ; Switch bank 0
*****************************************************************
* Main program
********@*****@*********************@**************************
        ...
*--------------------------------------------------------------
* Include required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\pitfall\tms9900\pitfall1a.a99"
        COPY  "D:\Projekte\pitfall\tms9900\pitfall1c.a99"
        ...
```

Notice the inclusion of **pitfall_bank_equates.a99** ? That is really important, more on that in the next topic.

Dependencies across banks

The biggest problem one faces during the implementation of a multi-bank ROM image, is having to deal with broken dependencies because of code and data realignment issues.

Suppose you are in subroutine SUBC in BANK 2 and want to jump to subroutine SUBB in BANK 1. Normally the steps you would do are:

- ➔ Get the address of SUBB in BANK 1.
- ➔ Set this address in subroutine SUBC in BANK 2.
- ➔ Assemble BANK 2

So far so good. Your program runs ok and your bank-switching code works just fine. You are a happy camper.

Then a few days later you need to fix a small bug in subroutine SUBA in BANK 1. You just had to add an "AI" statement and reassemble BANK 1. All seems to be ok at first. But then your program crashes after running subroutine SUBC in BANK 2. The problem is that your code still jumps to >6030 in BANK 1 but subroutine SUBB is not longer located there, it is now at >6034.

BANK 1 (BEFORE)

```
>6014    SUBA
            CLR R0
            MOV R1,R2
            RT
>6030    SUBB
            CLR R0
            MOV R1,R2
            ...
```

BANK 1 (AFTER)

```
>6014    SUBA
            CLR R0
            MOV R1,R2
            AI  R1,5
            RT
>6034    SUBB
            CLR R0
            MOV R1,R2
            ...
```

BANK 2

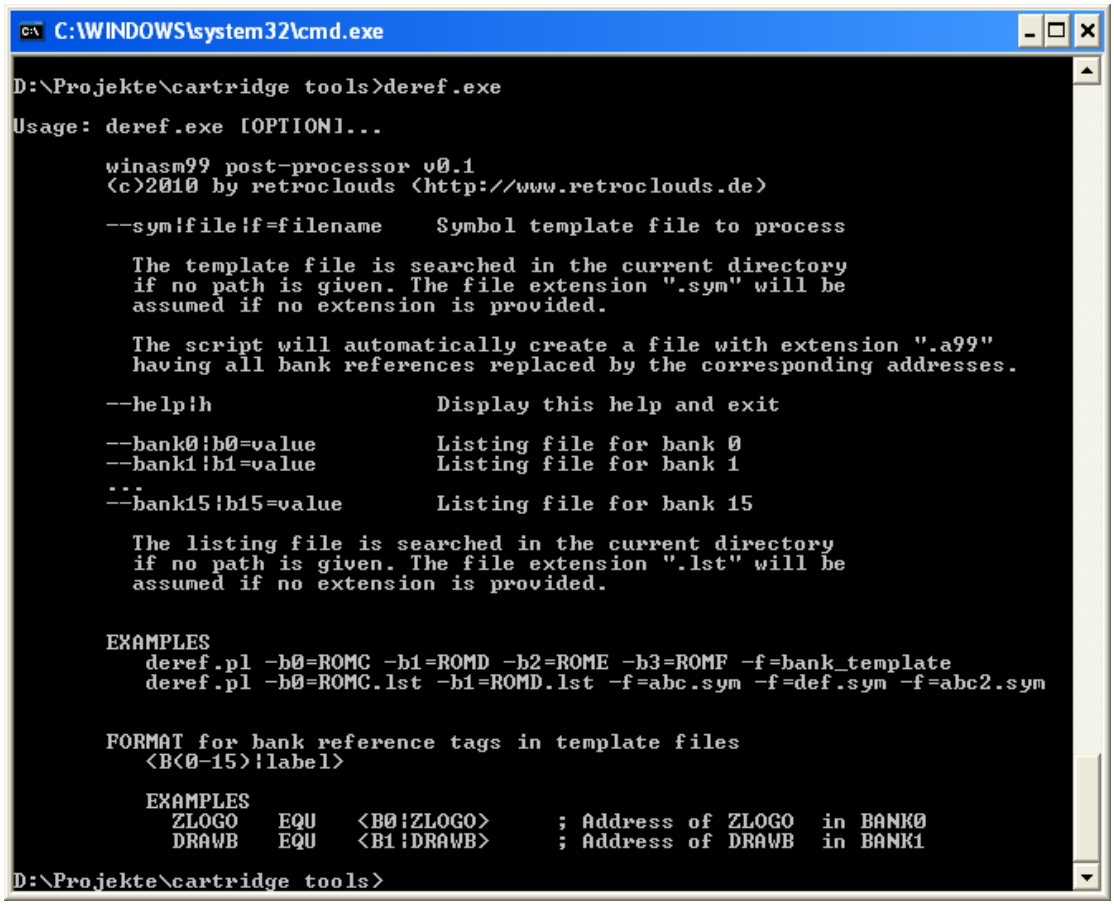
```
>6014    SUBC
            LI R1,>6002
            LI R2,>6030
            B  @GOBANK
            ...
```

To resolve the problem BANK 2 needs to be reassembled after adjusting the address in SUBC. Obviously these type of errors can easily result in a debugging nightmare. Updating all the addresses manually is almost an impossible task.

For handling this type of problem I wrote the command-line utility **deref.exe**

It basically scans the list files that get generated by winasm99 during the assembly process. Next it processes the specified template file, replacing the special tags with the corresponding ROM addresses and generate a matching assembly source file.

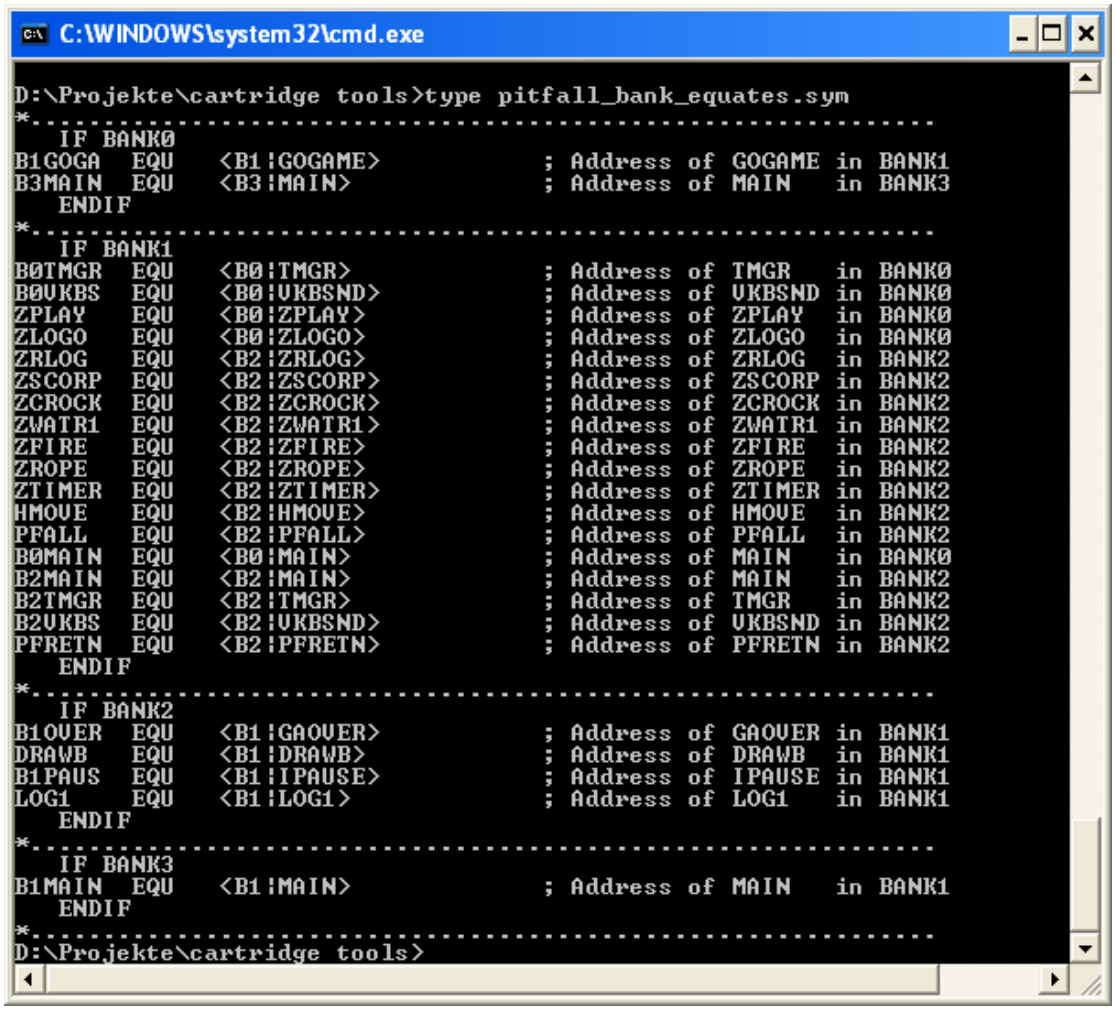
The command-line utility deref.exe



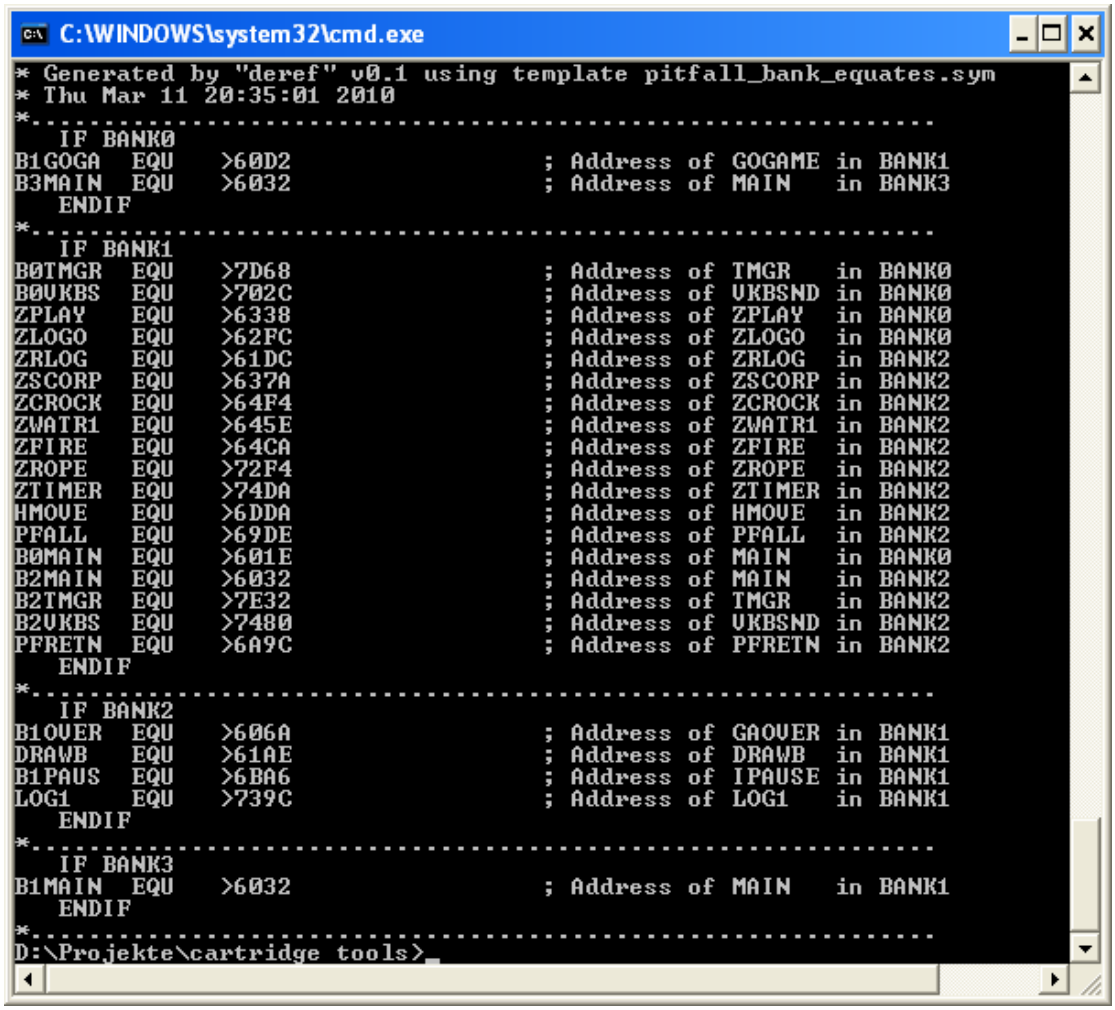
The idea is that you manually run deref.exe command-line utility after winasm99 has finished building a bank. It will generate a new assembly source with the updated ROM addresses. You can then run winasm99 for building the next bank.
This is a repetitive process that –depending on the source code modifications- needs to be done multiple times until all pieces fit together.

A template file is basically a plain text file. The only difference compared to normal assembly source files, are the special tags in the format **<B?|label>** where ? is the value of the bank number starting with 0.

The template file "pitfall_bank_equates.sym" used for building the Pitfall! ROM image.



The resulting **“pitfall_bank_equates.a99”** output file after running the deref.exe command-line utility.

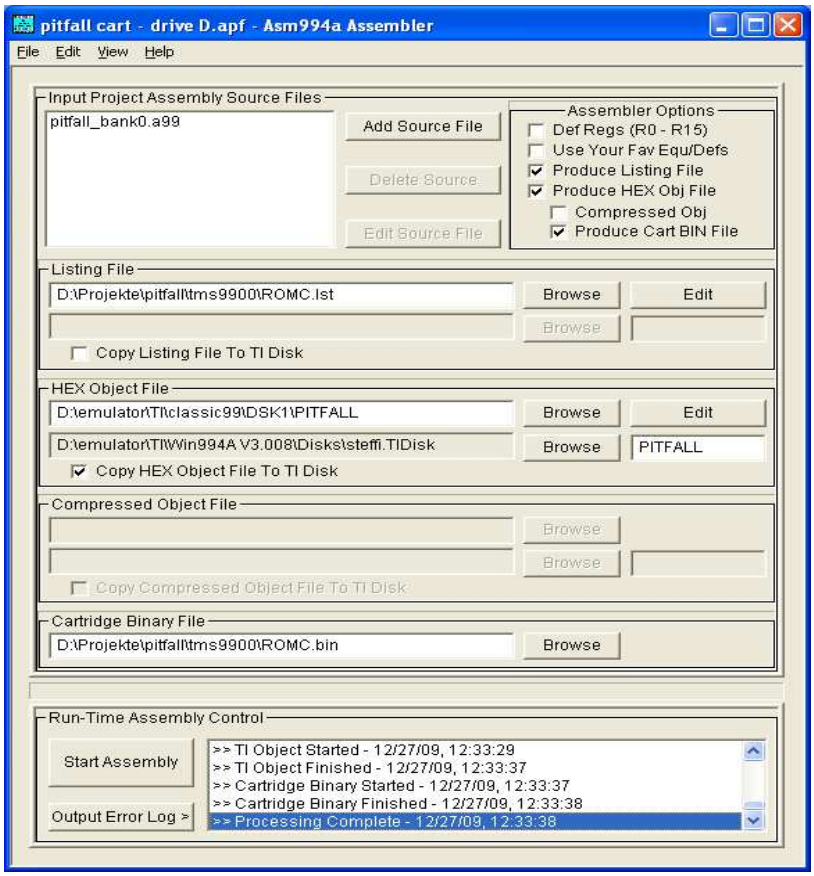


Building the multi-bank ROM image

Now let’s get on with the fun part; building the multi-bank ROM image. Assuming you have already unzipped the Pitfall! source directory to a local drive/device, you have to do the below steps for successfully creating the cartridge version.

Step 1: Open the files “pitfall_bank0.a99”, “pitfall_bank1.a99”, “pitfall_bank2.a99, “pitfall_bank3.a99” into your text editor and for each file replace the paths in all COPY statements so that it matches with your local source directory.
Your editors’ search & replace functionality will be of great use.

Step 2: After you have saved your modifications, start winasm99 and create a new project file with the settings seen in the next screenshot and click “Start assembly”. We are building BANK 0.



Step 3: Check for assembly errors. If all is fine you can then run the deref.exe command-line utility with the below parameters:

```
deref.exe -b0=ROMC -b1=ROMD -b2=ROME -b3=ROMF -f=pitfall_bank_equates.sym
```

A new “pitfall_bank_equates.a99” will be generated.

Step 4: Repeat the steps 2 & 3 for the other banks. The best way to do this is to run 4 parallel sessions of winasm99, one for each bank.

Below are the required settings for winasm99

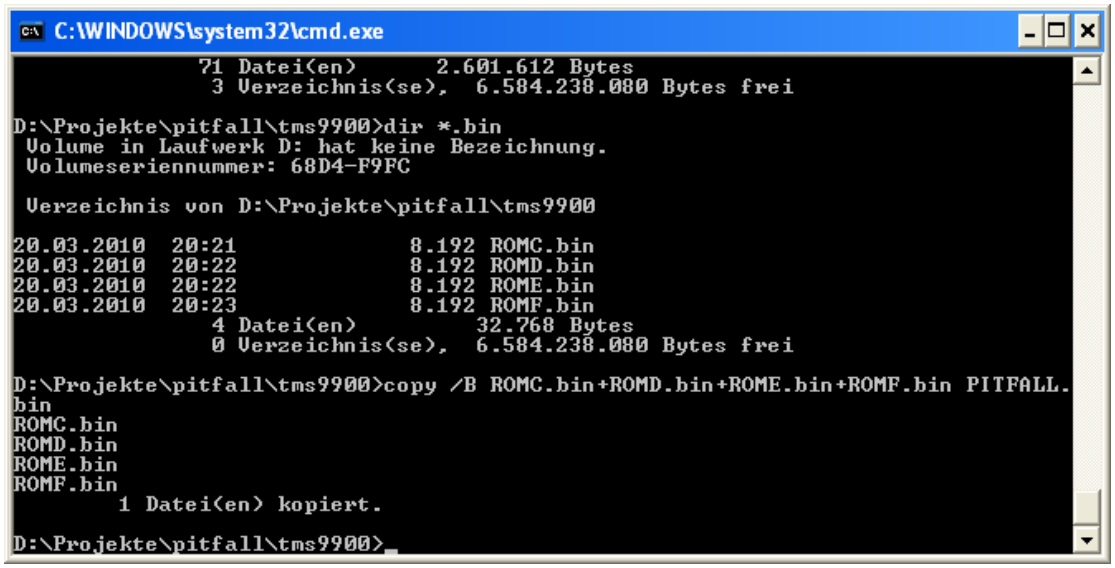
BANK	Input Project	Listing File	Cart binary file
BANK 0	pitfall_bank0.a99	ROMC.lst	ROMC.bin
BANK 1	pitfall_bank1.a99	ROMD.lst	ROMD.bin
BANK 2	pitfall_bank2.a99	ROME.lst	ROME.bin
BANK 3	pitfall_bank3.a99	ROMF.lst	ROMF.bin

Important! When modifying the source code you'll have to repeat the cycle of building banks 0-3 at least 2 times in a row until everything falls into place.
Best thing to do is monitor "pitfall_bank_equates.a99" after each run of the deref.pl script until there are no more changes from version to version.

Step 5: We are almost done. Now concatenate the 4 binary files ROMC.bin, ROMD.bin, ROME.bin, ROMF.bin for getting a single binary file PITFALL.bin

This is real easy when using the windows command window just type the below command:

```
Copy /B ROMC.bin+ROMD.bin+ROME.bin+ROMF.bin PITFALL.bin
```



Step 6: Congratulations, you have now successfully built a 32K ROM image. Enjoy!

Revision

Date	Author	Remark
10.01.2010	retroclouds	Initial version created (v0.1)
20.03.2010	retroclouds	Removed PERL and cygwin stuff, we now have an executable file (v0.2)