How to assemble Pitfall!

Introduction

This document concludes my work on one of my dearest projects of the last 2 years: The implementation of Pitfall! revision B, the arcade game for the Texas Instruments TI-99/4A Home Computer. This document describes the steps involved for building both the disk and cartridge version, starting from the same TMS9900 assembler source code.

Requirements

In order to build Pitfall! You need a recent version of the windows based cross–assembler **winasm99** which is delivered as part of the win994A emulator package[1]. I used winasm99 v3.8
Note that it may assemble with a newer version of winasm99 or other cross-assemblers targeting the TMS9900. The latter has not been tested by me and most likely there will be quite some manual steps involved in accomplishing this task.

Should you not only want to assemble the cartridge version, but also make modifications that require a relocation of code or a change in code size, then you will need to be able to run a PERL interpreter. Also you will have to find a way to concatenate the 4 x 8K binary files that build the 32K rom image.
I used the cygwin package[2] (UNIX like environment for windows) to accomplish these tasks.

All source code editing was done using the windows software notepad++[3]. A powerful text-editor that allows code-folding and search/replace across documents. The Pitfall! source code supports code-folding if you use my notepad++ template file[4].

Reference

[1] http://www.99er.net/win994a.shtml
[2] http://www.cygwin.com
[3] http://notepad-plus.sourceforge.net
[4] http://www.retroclouds.de

Source code details

The same source is used for building both the disk and cartridge version. The cartridge version will only differ from the disk version where necessary. I found out that winasm99 supports conditional defines, a feature which I believe is not officially documented (nor supported?). In each main file, I defined build directives as seen below.

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

In the below example the source code between ‘IF BANK1’ and ‘ENDIF’ is only assembled if the BANK1 equate equals 1. So I can include the same source file in multiple projects and control how it gets assembled by setting the corresponding equates to 1 or 0.

Note that what you see in the example below, is a bank-switch from BANK1 (2nd bank) to BANK2 (3rd bank)

```
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
```

Build the disk version

Building the disk version is the easiest part as you don’t have to worry about bank-switching and code layout. Note that if -in the future- a sideport-cartridge gets implemented with 24K address space in high memory >A000, it will be a snap to assemble the disk version for that purpose. We are only using scratch-pad memory so we should be fine.

Let’s get started

Assuming you have already unzipped the Pitfall! source directory to a local drive/device, you have to do the below steps for successfully creating an E/A#3 object file.

Step 1: Open the file “pitfall_disk.a99” into your text editor and 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, as there are about 38 COPY statements to be adjusted.

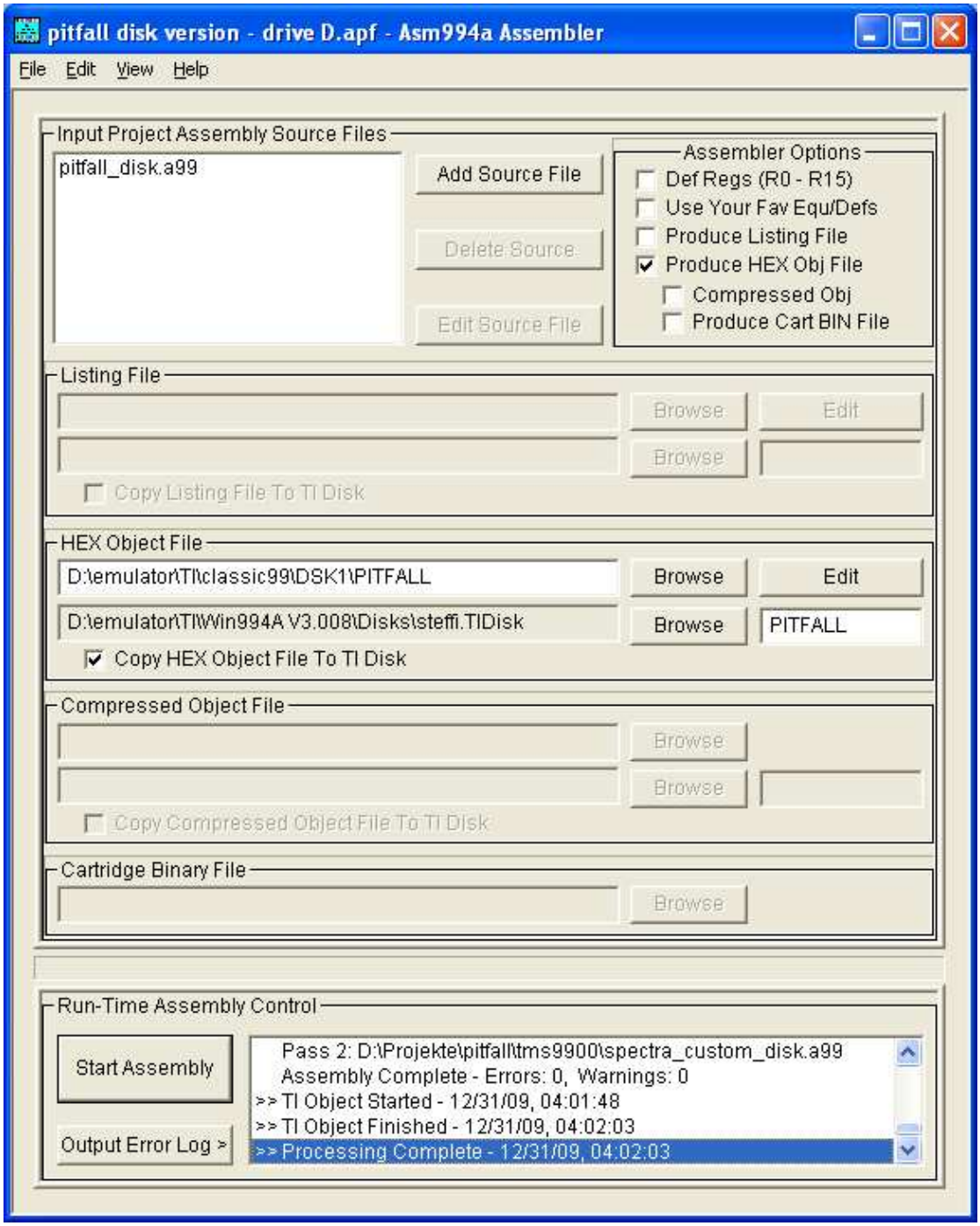
```
*--------------------------------------------------------------
* Include all required files
*--------------------------------------------------------------
        COPY  "D:\Projekte\pitfall\tms9900\pitfall1.a99"
        COPY  "D:\Projekte\pitfall\tms9900\pitfall1a.a99"
        COPY  "D:\Projekte\pitfall\tms9900\pitfall1b.a99"
        ...
```

Step 2: Depending if your planning on turning your fresh #EA3 object file into a #EA5 memory image file or not, you’ll have to jump to the bottom of the source file and uncomment/comment one of the last two lines as seen below.

```
*--------------------------------------------------------------
* End of Game
*--------------------------------------------------------------
SLAST
        END                         ; For use with SAVE utility
*       END   SFIRST                ; For autostart
```

The actual conversion to the #EA5 memory image file can later be done via the SAVE utility which is part of Texas Instruments Editor/Assembler package.

Step 3: After you have saved your modifications, start winasm99 and create a new project file with the settings seen in the next screenshot. Depending on your requirements you may want winasm99 to inject the generated #EA3 object file directly into a disk image, for trying with the MESS emulator. If you are using classic99 for testing/debugging, it is sufficient to let winasm99 store the generated #EA3 file in the DSK1 folder of the classic99 emulator.



Step 4: Check for assembly errors. Congratulations if all is fine, you have just successfully built the disk version.

Build the cartridge version

Now let's get on with the fun part; building the cartridge version. Before we can do that I'll explain how the code layout for the Pitfall! cartridge version looks like.

Code layout

The game is split in 4 banks of 8K each. I tried to reduce the number of times the game needs to switch banks to an absolute minimum. That had a big impact on the code layout.

Basically each bank is divided in two parts. Depending on the bank the first part is the actual game code and/or graphics and sound data. The second part of each bank contains a custom version of SPECTRA[1], my arcade game library for the TI-99/4A.
Mind you, the included SPECTRA code is NOT the same in all 4 banks. Only the subroutines and data required for supporting the game code of the bank it resides in is included. The obvious reason for that is to save space.

<table>
<tr><th>BANK 0</th><th>BANK 1</th><th>BANK 2</th><th>BANK 3</th></tr>
<tr><td>Cartridge header

TITLE SCREEN

ANIMATE HARRY

SWING ROPE

Scroll logo Horizontally

Scroll copy-right vertically

Graphics and sound data</td><td>Cartridge Header

DRAW SCENERY

GAME START

GAME OVER

PAUSE FUNCTION

SETUP ENEMIES</td><td>Cartridge header

ANIMATE/CONTROL HARRY

ANIMATE/CONTROL ENEMIES

GAMEPLAY & SCORE

SWING ROPE

Collision Detection</td><td>Cartridge Header

Setup all graphics/sound data in VDP.

Graphics and sound data

255 Levels</td></tr>
<tr><td>>79D6 - >7FF4
SPECTRA
1554 bytes</td><td>>79E2 - >7FFF
SPECTRA
1566 bytes</td><td>>7BBA - >7FFF
SPECTRA
1094 bytes</td><td>>7C0C - >7FFF
SPECTRA
1012 bytes</td></tr>
<tr><td>ROMC.bin</td><td>ROMD.bin</td><td>ROME.bin</td><td>ROMF.bin</td></tr>
</table>

Note: The bank-switching code resides at >79D6, >79E2, >7BBA, >7C0C just before SPECTRA.

The cartridge header is present in each bank, as the TI-99/4A can fire up a random bank on reset. In each bank it points to a routine that switches to BANK 0 and triggers the initialisation routine.

Bank Switching sequences

Below you find the bank-switching sequences used in Pitfall!

Startup sequence:

BANK0 (TITLE SCREEN ANIMATION) -> BANK3 (SETUP GRAPHICS/SOUND) -> BANK1 (DRAW SCENERY) -> BANK0 (SCROLL ACTIVISION COPYRIGHT while waiting for game start) -> BANK1 (START GAME) -> BANK2 (PLAY GAME)

Switching to new screen:

BANK2 (PLAY GAME) -> BANK1 (DRAW SCENERY) -> BANK2 (PLAY GAME)

Pause game:

BANK2 (PLAY GAME) -> BANK1 (BLINK PAUSE and wait until key is pressed) -> BANK2 (PLAY GAME)

GAME OVER:

BANK2 (PLAY GAME) -> BANK1 (GAME OVER) -> BANK0 (SCROLL ACTIVISION COPYRIGHT while waiting for game start)

GAME START:

BANK3 (SETUP GRAPHICS/SOUND) -> BANK1 (DRAW SCENERY) -> BANK0 (SCROLL ACTIVISION COPYRIGHT while waiting for game start) -> BANK1 (START GAME)-> BANK2 (PLAY GAME)

Dependencies across banks

One of the biggest problems faced during the development of the cartridge version, is dealing with broken dependencies due to code & ROM data reallocation during the development/debugging phase.
Some routines depend on routines that are located in another bank.
That means that if you, for example add an assembly instruction in BANK1 and reassemble, everything will be fine in BANK1. However you will have invalid references in BANK2, as they are still pointing to the old addresses. Updating the addresses manually is almost an impossible task. Therefore the below solution was implemented.

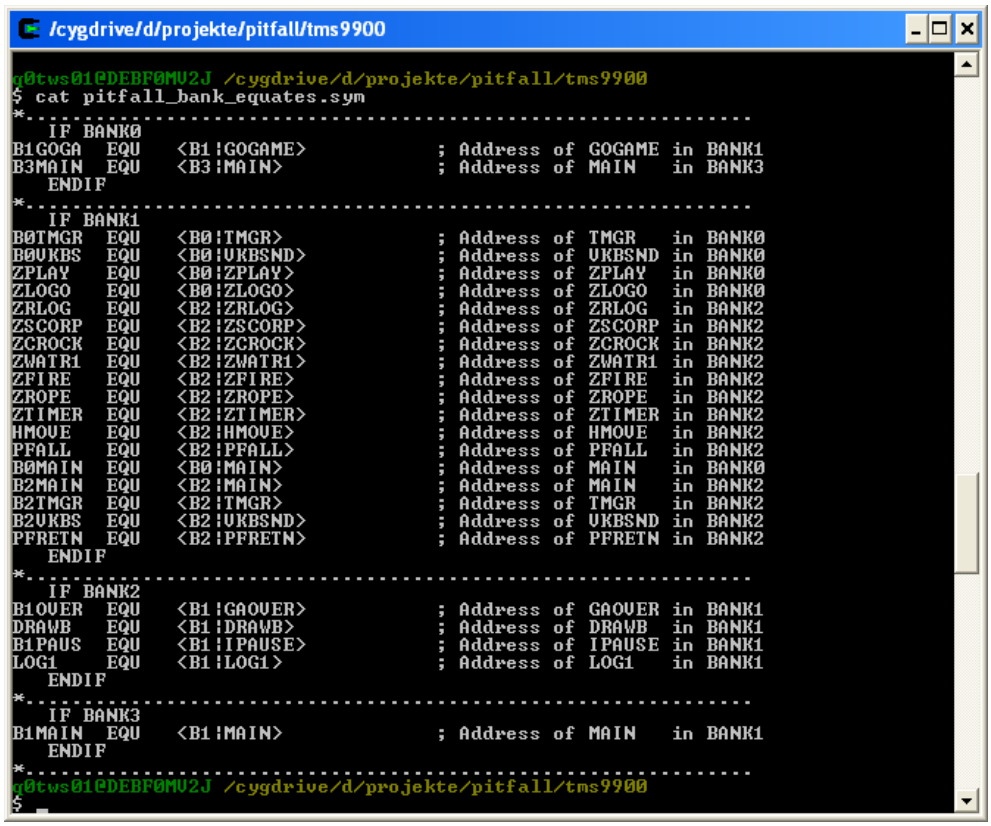
I developed a PERL script called **merger** that reads the 4 list files (ROMC.lst, ROMD.lst, ROME.lst, ROMF.lst) which get generated by winasm99 while assembling the bank images.

The script has three major tasks

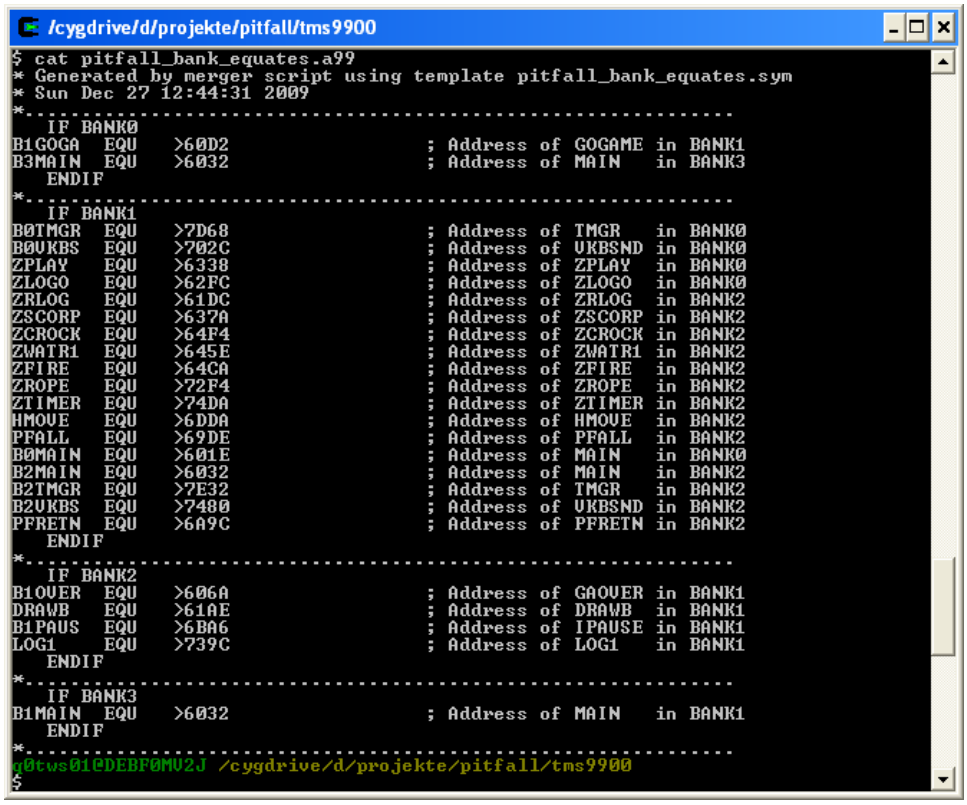
- ➔ Check how much space is left in each bank image.
- ➔ Warn if code gets overwritten due to AORG statements.
- ➔ Proces the custom file “pitfall_bank_equates.sym”, replacing the special tags with the corresponding ROM addresses and generate “pitfall_bank_equates.a99”

The input file “**pitfall_bank_equates.sym**” has special tags in the format **<B?|label>** where ? is the value of the bank number starting with 0.

“**pitfall_bank_equates.sym**” input file for merger script.



“**pitfall_bank_equates.a99**” output file created by the merger script.



The file “pitfall_bank_equates.a99” is included via a COPY directive in each main file used for building the bank. The idea is that you manually run the merger script after winasm99 has finished. This is a repetitive process that you need to cycle through at least two times before all pieces fit together. However, it will save you a considerable amount of time.

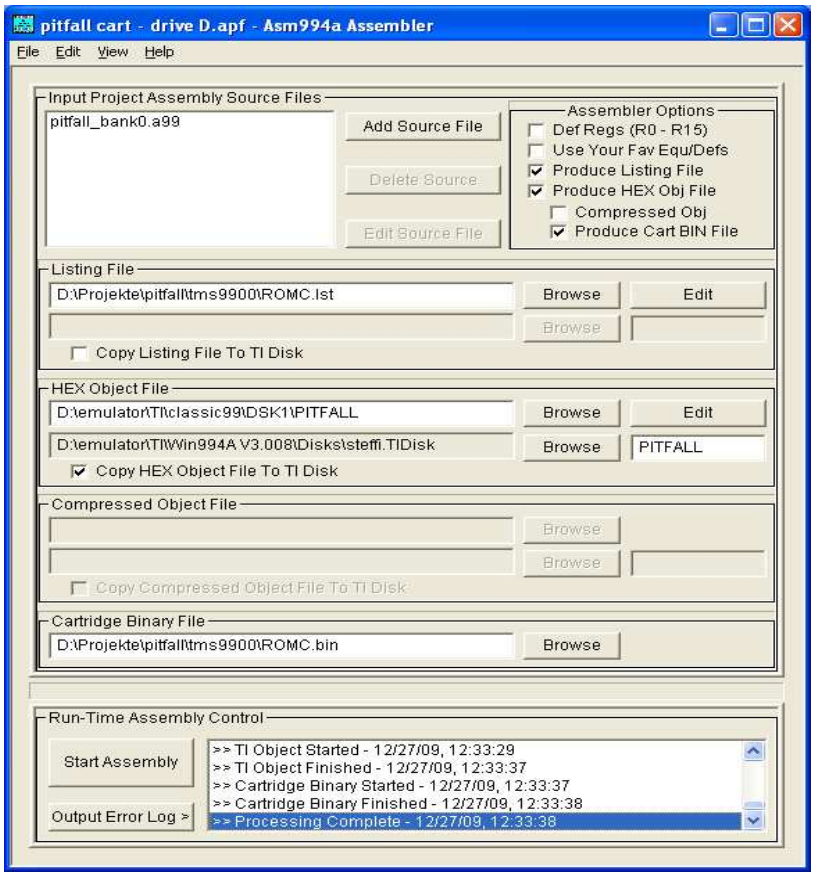
Let’s get started

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

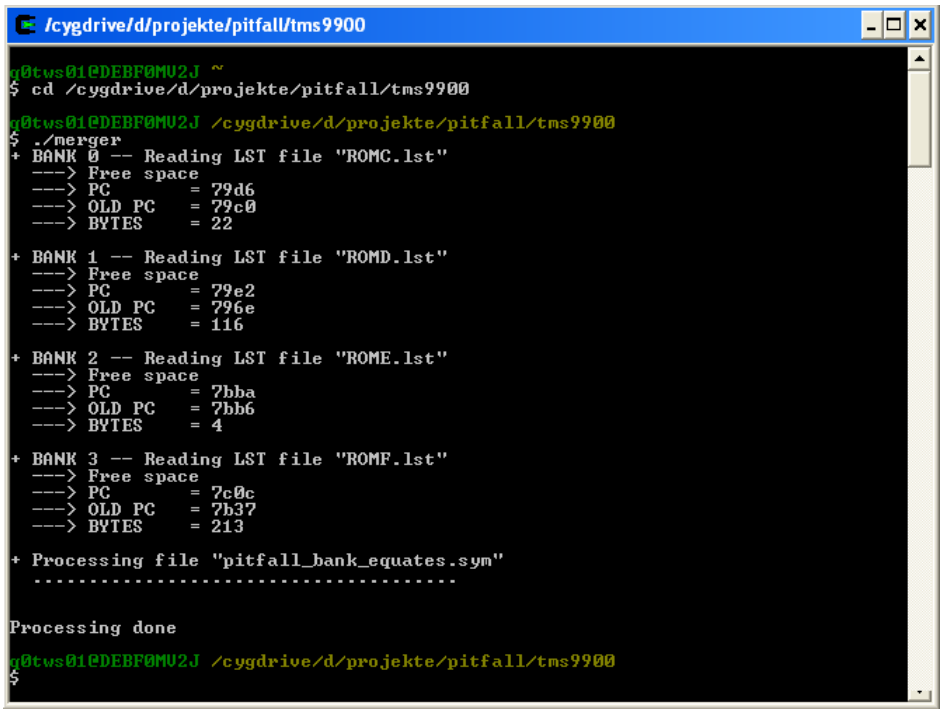
Step 1: Open the files “pitfall_bank0.a99”, “pitfall_bank1.a99”, “pitfall_bank2.a99, “pitfall_bank3.a99” into your text editor and for each file replace the paths in all COPY statements so that it matches with your local source directory.

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 “merger” script.



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.

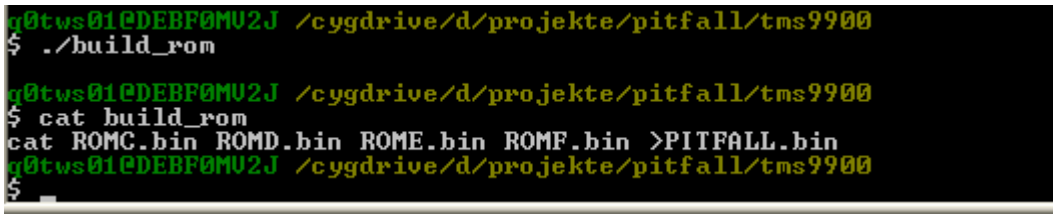
Below are the most important settings

Input Project	Listing File	Cartridge binary file
pitfall_bank0.a99	ROMC.lst	ROMC.bin
pitfall_bank1.a99	ROMD.lst	ROMD.bin
pitfall_bank2.a99	ROME.lst	ROME.bin
pitfall_bank3.a99	ROMF.lst	ROMF.bin

Important! Due to cross-dependencies you must repeat the cycle of building banks 0-3 at least 2 times in a row until everything falls into place.
Best is to monitor "pitfall_bank_equates.a99" after each run of the merger 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 cygwin, just run my script "build_rom".



Step 6: Congratulations, you have successfully built the cartridge version. Enjoy!

Revision

Date	Author	Remark
01.01.2010	retroclouds	Initial version created