Cross compiling to 6502 8-bit systems with 'cc65'

Bill Kendrick
New Breed Software

Linux Users' Group of Davis
September 14, 2015

6502, a 40 year old CPU still in use!

- An 8-bit microprocessor
- Released in 1975 by MOS Technology
  - MOS = Metal Oxide Semiconductor
  - Former Motorola employees moved to MOS to produce a low-cost CPU (alt. to 6800)
  - Lots of history:
    - https://en.wikipedia.org/wiki/MOS_Technology_6502
    - http://6502.org/
- I'm not a CPU expert or historian; I'm sure some of you here know *way* more than me <looks at Steve>

Atari VCS...

- Atari, founded in 1972, began with arcade games (Pong, Space War, etc.)
- 1977 the Video Computer System (VCS, later known as 2600) was released
  - 6502-based home video game console
  - Cartridge-based (vs. built-in) games
    - (2nd to do so, but arguably industry-changer)
  - Inadverdently created 3rd-party game software industry
  - RF-based output, TV-oriented video chip (TIA, Television Interface Adapter)
  - Whopping 128 *bytes* of RAM; cart. ROMs up to 4K (w/o tricks)
  - https://en.wikipedia.org/wiki/Atari_2600

Atari 8-bits, the next VCS/2600?

- 1979, Atari 400 & 800 released
  - 6502-based home computer; up to 48KB RAM, 10KB OS ROM, BASIC on cartridge
  - “CTIA” or “GTIA” video-driving graphics chip
  - “ANTIC” video coprocessor, w/ own instruction set
  - “POKEY” I/O chip (sound, keyboard, analog-to-digital)
  - “SIO” plug-n-play, daisy-chain'able serial bus interface
    - Not as fast as parallel; designed to reduce RF output (FCC regulations; see also: aluminum encasing)
    - Disk drives, cassette drives, printers, MODEMs, etc.
    - Joe Decuir, who holds patents for SIO, worked on USB!
  - ROM-based cartridge software, too
  - https://en.wikipedia.org/wiki/Atari_8-bit_family

- 1980s, XL and XE series of home computers (64-128KB RAM, 16KB ROM, etc.)
- 1982, Atari 5200 SuperSystem, based on 6502, GTIA, ANTIC & POKEY
  - https://en.wikipedia.org/wiki/Atari_5200 ← so... yes :)

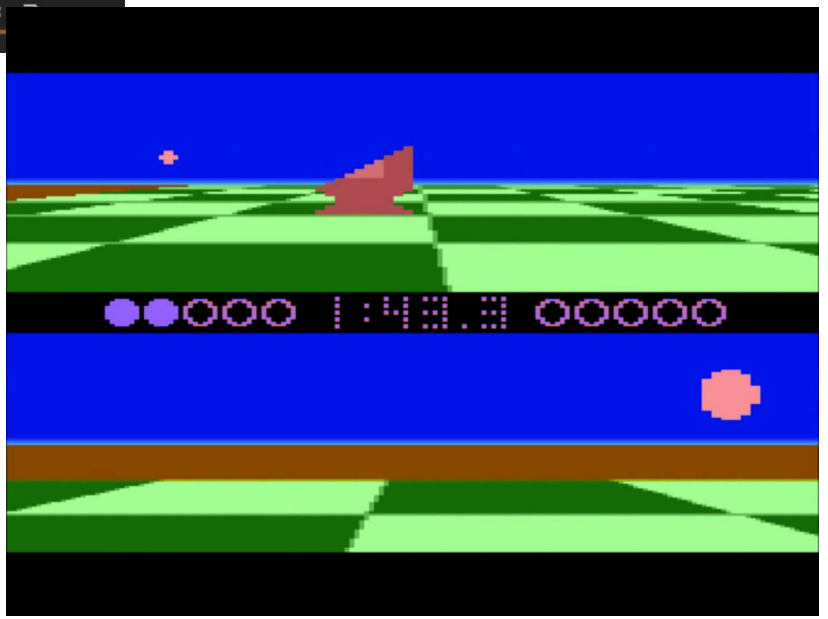
FFWD - predecessor to the Amiga!

- Jay Miner, who worked on 2600 TIA and 400/800 CTIA & ANTIC, wanted to create a 16-bit, floppy-disk-based game console
  - “It's complicated” https://en.wikipedia.org/wiki/Amiga_Corporation http://lowendmac.com/orchard/06/amiga-origin-commodore.html
- ANTIC + GTIA provided:
  - 40 column, 24 row 2-color text
  - 20x12 & 20x24 colored text modes (5 colors total)
  - 40x12 & 40x24 multicolored text modes (4 colors per char.)
  - 320x192 2-color high res. graphics
  - 160x192, 160x96, 80x96 & 40x48 4-color graphics
  - 128 colors (16 hues x 16 shades), mapped to 9 palette registers (possible to get 256, too)
  - Redefinable character sets (aka fonts; also useful for tile-based graphics)
  - “Player/Missile Graphics”, aka “sprites”
  - Hardware horizontal & vertical fine scrolling

More on Atari graphics

- Normal, narrow (thick borders), and wide (overscan) playfield modes
- ANTIC's instruction set define what's on the screen:
  - What graphics mode? Where in RAM to fetch from? Enable scrolling? Call DLI? (below)
- Vertical Blank Interrupts (VBI) – mainline 6502 code is interrupted...
  - Code that runs during while the CRT's electron beam returns to top left corner, to start a new frame (reasonable amount of time for code to run, and runs regularly, 60x/second (NTSC))
- Display List Interrupts (DLI) – mainline 6502 code is interrupted...
  - Code that runs while beam returns to left side, as it scans (not very much time for code, but happens ~12,000x/second)
  - Useful for adjusting GTIA chip's registers, to change graphics at points down the screen (e.g., player/missile position, color registers)
- VBI & DLI are useful for fast I/O
  - Music & sfx playback. Reading mouse or Trak-Ball input device. Etc

Atari 8-bit Graphic Examples (old & new)



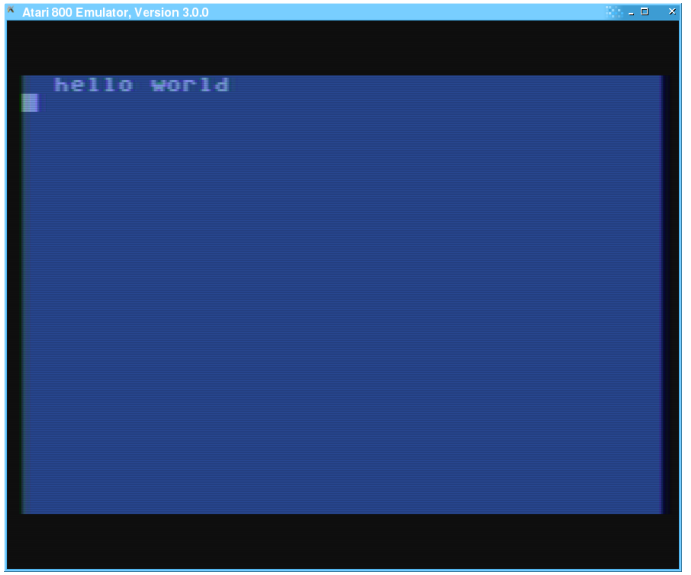
cc65; an Atari C compiler

- 1989(?), cc65 created for the Atari 8-bit (written using MAC/65 assembler on an Atari)
  - cc65: compile C source to assembly
  - ra65: assemble that into object file
  - link65: link object with C runtime, build executable
  - http://www.umich.edu/~archive/atari/8bit/Languages/Cc65/
- 1997(?), ported to UNIX
- 1999, development continued...

cc65: a cross-platform compiler

- Today, compiles with GNU *gcc*, runs on:
  - Linux, Windows, Mac OS X, etc.
- Targets various 6502-based platforms:
  - Apple II
  - Atari 2600, 8-bit computers, 5200, Lynx handheld
  - Commodore computers (C=64, C=128, VIC20...)
  - Nintendo Entertainment System (NES)
  - ...and more
- (new) Home: http://cc65.github.io/cc65/
- Wiki: https://github.com/cc65/wiki/wiki

Hello, world



- hello.c:

```
#include <stdio.h>
#include <unistd.h>
int main(void) {
  printf("hello world\n");
  sleep(2);
  return(0);
}
```

- Compile, assemble & link (one fell swoop, with “cl65”):

```
$ export CC65_HOME=/usr/local/share/cc65/
$ cl65 -t atari hello.c -o hello.xex
```

- Run in emulator:

```
$ atari800 -run hello.xex
```

Building one step at a time

- Compile C to assembly (.c → .s)
```
$ export CC65_HOME=/usr/local/share/cc65/
$ cc65 hello.c
```
- Assemble assembly to object (.s → .o)
```
$ ca65 hello.s
```
- Link object & runtime to executable (.o & .lib → .xex)
```
$ ld65 hello.o atari.lib -t atari -o hello.xex
```

Peering inside (assembly)

```
.segment        "RODATA"

L0003:
        .byte   $68,$65,$6C,$6C,
$6F,$20,$77,$6F,$72,$6C,$64,$0A,
$00

; int __near__ main (void)

.segment        "CODE"
```

```
.proc   _main: near
.segment        "CODE"
        lda     #<(L0003)
        ldx     #>(L0003)
        jsr     pushax
        ldy     #$02
        jsr     _printf
        ldx     #$00
        lda     #$02
        jsr     _sleep
        ldx     #$00
        lda     #$00
        jmp     L0001
L0001:  rts
.endproc
```

Peering inside (mapfile)

- Mapfiles contain a detailed overview of the modules used, the sizes for the different segments, and a table containing exported symbols

```
$ ld65 hello.o -t atari \
  -o hello.xex atari.lib \
  --mapfile hello.map
```

Character Sets on Atari

```
128  64  32  16   8    4   2   1
...---.....---.....---.....---- =   0
...---....########---.....---- =  24
...----################.....---- =  60
....########-------########---- = 102
....########-------########---- = 102
....########################---- = 126
....########-------########---- = 102
...---.....---.....---.....---- =   0
```

- Address 756 (0x2F4) used by OS, “Character Base Register”, where ANTIC accesses (via DMA, Direct Memory Access) the values to render in text modes
- Points to a 'page' (sections of 256 (0x100) bytes of memory) where 1KB of font data is stored
  - 128 characters x 8 bytes per character

Character Sets on Atari

- In BASIC:

```
10 MEMTOP=PEEK(106)
20 CHSET=MEMTOP-8:REM 4 pages = 1KB
30 POKE MEMTOP,CHSET
40 GRAPHICS 0
50 POKE 756,CHSET
60 CHSET=CHSET*256:REM pages->bytes
70 FOR I=0 TO 255
80 POKE CHSET+I,I
90 PRINT I;
100 NEXT I
```

Memory Configuration

- cc65 toolset requires a memory configuration file to define the memory that is available to the cc65 run-time environment
  - http://cc65.github.io/doc/customizing.html#s2
  - e.g. `/usr/local/share/cc65/cfg/atari.cfg`
- Let's make room for a font:

```
FONT: load = RAM, type = rw, define = yes align=$1000;
```

- Include the font, in that location:

```
#pragma data-name (push,"FONT")
#include "font.h"
#pragma data-name (pop)
```

Switch to the font

- ```
  #define CHBAS *(unsigned char *) 0x2F4
  CHBAS = ((unsigned int) &font)/256;
  ```
- Alternatively,
  ```
  #include <peekpoke.h>
  POKE(756, ((unsigned int) &font)/256);
  ```
  ```

Let's Copy That Floppy!

- ATR floppy disk image format
  - Used by emulators (atari800, etc.)
  - Used with disk simulation cables & PC apps (SIO2PC, AspeQt, etc.) & stand-alone devices (SIO2SD, etc.)
  - Used with disk-emulating cartridges (MaxFlash, The!CART)
- “Franny”, open source tool to manipulate Atari disk images (.atr) (there are other tools)
  - Part of “atari8” open source project http://atari8.sourceforge.net/

Making a Disk

- `franny -C mydisk.atr -d s -f a`
  - -C → create
  - -d s → sector size: single density
  - -f a → filesystem type: Atari DOS 2.x
- `franny -F mydisk.atr`
  - -F → format
- `franny -A mydisk.atr -i hello.xex -o HELLO.EXE`
  - -A → add file
  - -i → input (source) local file
  - -o → output (destination) filename in disk image

But that's not bootable :(

- `atari800 bootable.atr mydisk.atr`
- Run your “HELLO.EXE” off of drive 2, e.g. in Atari DOS or MyDOS:
  - [L]oad Memory
  - `D2:HELLO.EXE` [Return]

Bill's Crazy Hack to Make Boot Disk

- Take a MyDOS bootable disk image
- Use “xxd(1)” to extract the first 3 sectors into a file
  - Single density: 128 x 3 = 384 bytes (0x180)
- Use Franny or an emulator (e.g., atari800's “H:” device to read/write to host filesystem) to extract “DOS.SYS” (bootable)
- Insert this step after “franny -F” to format your new disk image:
  - `( cat first3sectors.xxd ; xxd -s 0x180 mydisk.atr ) \ | xxd -r > mydisk2.atr`
    - -s → seek (skip 384 bytes)
    - -r → revert (from the combined cat & xxd dumps, back into a binary file)
- “`franny … -O AUTORUN.SYS`” (Atari DOS)
  or “`franny … -O HELLO.AR0`” (MyDOS)

Some room to load

- In the .cfg file:
```
GFX:  load = RAM,  type = rw,  define = yes;
```
- In the .c source:
```
#pragma bss-name ("GFX")
unsigned char gfx[1024];
```
- (see hello.3)
- The empty space *isn't* stored in the executable file!

Why?

- Easier & faster development
  - Makefiles, fast compiler, turbo mode in emulator
- C is well known & powerful
  - BASICs are slow & less useful
  - Action!/etc. are relatively uncommon
  - Straight assember is hard :-P
  - Structs & functions are useful
- Can code on laptop in livingroom near family
  - Harder to do so with an Atari & CRT monitor :-)