PROJECT DESIGN:

1802 Microprocessor on Altera Cyclone V SoC

Prepared by Jennifer Bi,
Nelson Gomez,
Kundan Guha,
and Justin Wong

Embedded Systems 4840

March 30, 2019

Contents

1 Introduction **5**
- 1.1 Purpose . . . 5
- 1.2 References . . . 5

2 System Design Overview **6**
- 2.1 User Interfaces . . . 6
- 2.2 Hardware-Software Interfaces . . . 6

3 CPU **7**

4 Memory Overview **8**

5 Peripherals **9**
- 5.1 Graphics . . . 9
- 5.2 Sound . . . 9

6 ISA **10**
- 6.1 1802 ISA . . . 10
- 6.2 00:IDL . . . 11
- 6.3 0N: (except N=0) Load . . . 11
- 6.4 1N: INC . . . 11
- 6.5 2N: DEC . . . 11
- 6.6 30: BR . . . 11
- 6.7 31: BQ . . . 11
- 6.8 32: BZ . . . 11
- 6.9 33: BDF . . . 11
- 6.10 34: B1 . . . 11
- 6.11 35: B2 . . . 11
- 6.12 36: B3 . . . 12
- 6.13 38: SKP . . . 12
- 6.14 39: BNQ . . . 12
- 6.15 3A: BNZ . . . 12
- 6.16 3B: BNF . . . 12
- 6.17 3C: BN1 . . . 12
- 6.18 3D: BN2 . . . 12
- 6.19 3E: BN3 . . . 12
- 6.20 3F: BN4 . . . 12
- 6.21 4N: LDA . . . 12

6.22 5N: STR . . . 13
6.23 60: IRX . . . 13
6.24 61-67 . . . 13
6.25 68 . . . 13
6.26 69-6F . . . 13
6.27 70 . . . 13
6.28 71 . . . 13
6.29 72: LDXA . . . 13
6.30 73: STXD . . . 13
6.31 74: ADC . . . 13
6.32 75: SDB . . . 14
6.33 76:SHRC . . . 14
6.34 77: SMB . . . 14
6.35 78: SAV . . . 14
6.36 79: MARK . . . 14
6.37 7A: REQ . . . 14
6.38 7B: SEQ . . . 14
6.39 7D: SDBI . . . 14
6.40 7E:SHLC . . . 14
6.41 7F: SMBI . . . 15
6.42 8N: GLO . . . 15
6.43 9N: GHI . . . 15
6.44 AN: PLO . . . 15
6.45 BN: PHI . . . 15
6.46 C0:LBR . . . 15
6.47 C1: LBQ . . . 15
6.48 C2: LBZ . . . 15
6.49 C3: LBDF . . . 15
6.50 C4: NOP . . . 15
6.51 C5: LNQ . . . 16
6.52 C6: LSNZ . . . 16
6.53 C7: LSNF . . . 16
6.54 C8: NLBR . . . 16
6.55 C9: LBNQ . . . 16
6.56 CA: LBNZ . . . 16
6.57 CB: LBNF . . . 16
6.58 CC: LSIE . . . 16
6.59 CD: LSQ . . . 16
6.60 CE: LSZ . . . 16
6.61 CF: LSDF . . . 17
6.62 DN: SEP . . . 17
6.63 EN: SEX . . . 17
6.64 F0 . . . 17
6.65 F1: OR . . . 17

6.66 F2: AND . . . 17
6.67 F3: XOR . . . 17
6.68 F4: XOR . . . 17
6.69 F5: SD . . . 17
6.70 F6: SHR . . . 17
6.71 F7: SM . . . 18
6.72 F8: LDI . . . 18
6.73 F9: ORI . . . 18
6.74 FA: ANI . . . 18
6.75 FB: XRI . . . 18
6.76 FC: ADI . . . 18
6.77 FD: SDI . . . 18
6.78 FE: SHL . . . 18
6.79 FF: SMI . . . 18
6.80 INTERRUPT . . . 19
6.81 DMA-IN . . . 19
6.82 DMA-OUT . . . 19

7 Interrupts and DMA **20**

8 Testing using Verilator **21**

1 Introduction

1.1 Purpose

Our goal is to implement the CPD1802 Microprocessor, along with the CPD1861 Video Display Controller, on the Altera Cyclone V SoC FPGA using System Verilog. We will remain faithful the original COSMAC specifications as much as possible. We will verify our 1802 implementation with a test suite written in Verilator.

We will also provide a small software interface on the SoC's ARM Hard Processor System for starting/stopping the 1802, loading memory, and sending keystrokes to the 1802. In particular, this will enable us to load and run programs, including the Chip8 interpreter and Chip8 games.

1.2 References

1. CDP1802 Microprocessor Specification: *http://www.cosmacelf.com/publications/data-sheets/cdp1802.pdf*

2. CDP1861 Video Display Controller: *http://www.cosmacelf.com/publications/data-sheets/cdp1861.pdf*

3. *http://www.cs.columbia.edu/ sedwards/classes/2016/4840-spring/designs/Chip8.pdf*

4. Complete CHIP-8 Interpreter Listing, which we can load onto our 1802: *http://cosmacelf.com/forumarchive/files/CHIP-8/ELF%20CHIP-8%20Interpreter.pdf*

5. *http://laurencescotford.co.uk/wp-content/uploads/2013/07/RCA1802-Instruction-Set.pdf*

6. http://bitsavers.trailing-edge.com/components/rca/cosmac/MPM-201A_User_Manual_for_the_CDP1802_COSMAC_Microprocessor_1976.pdf

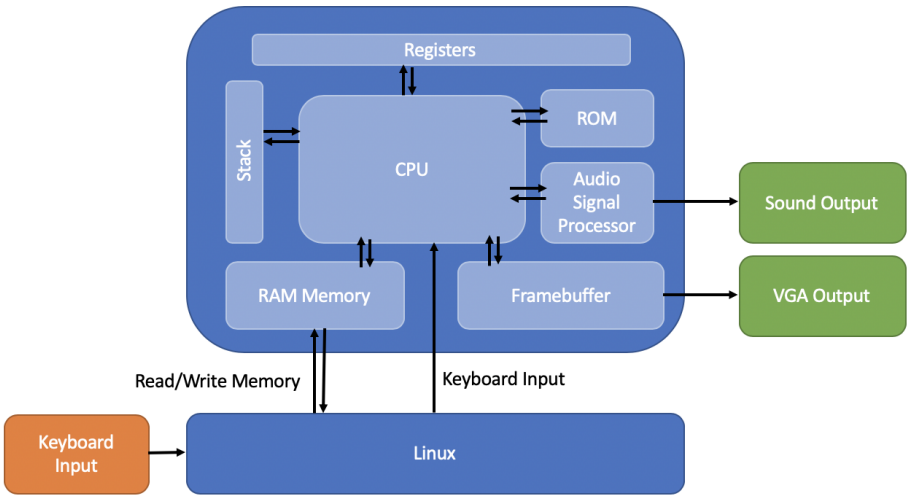
2 System Design Overview

2.1 User Interfaces

We allow the user to interact with the system by keyboard input to the ARM Linux. This input is then passed on to the 1802 hardware where the actual interrupt will be handled. Further, we reserve a key not relevant to 1802's execution from which we can pause and resume the execution of the hardware system.

2.2 Hardware-Software Interfaces

We utilize two separate 8-bit data buses between hardware and ARM Linux where we can communicate the key-presses and also read and write data into the memory space of 1802. The 1802 uses two latches (TPA and TPB), but we may decide to not use these and have bus timing be synchronous to the rising edge of the clock.



3 CPU

The 1802 has an array of sixteen 16-bit registers, and an accumulator D to hold to any intermediate computation results. Each CPU instruction uses two machine cycles: fetch and execute. During the fetch cycle, the 4-bit P register selects one of the sixteen registers as the current program counter, and the instruction is read out from memory using the PC/address. The 4-bit X register selects one of the sixteen register as the memory address to the operand for ALU or I/O operations. Registers can also store immediate data.

There are four control modes: `LOAD, RESET, PAUSE, RUN`. We will have a RUN switch in software, which will allow us to switch between RESET and RUN.

The CPU block diagram from the 1802 specification is shown below. Timing diagrams from the spec will also be useful to us, but we won't show them here.

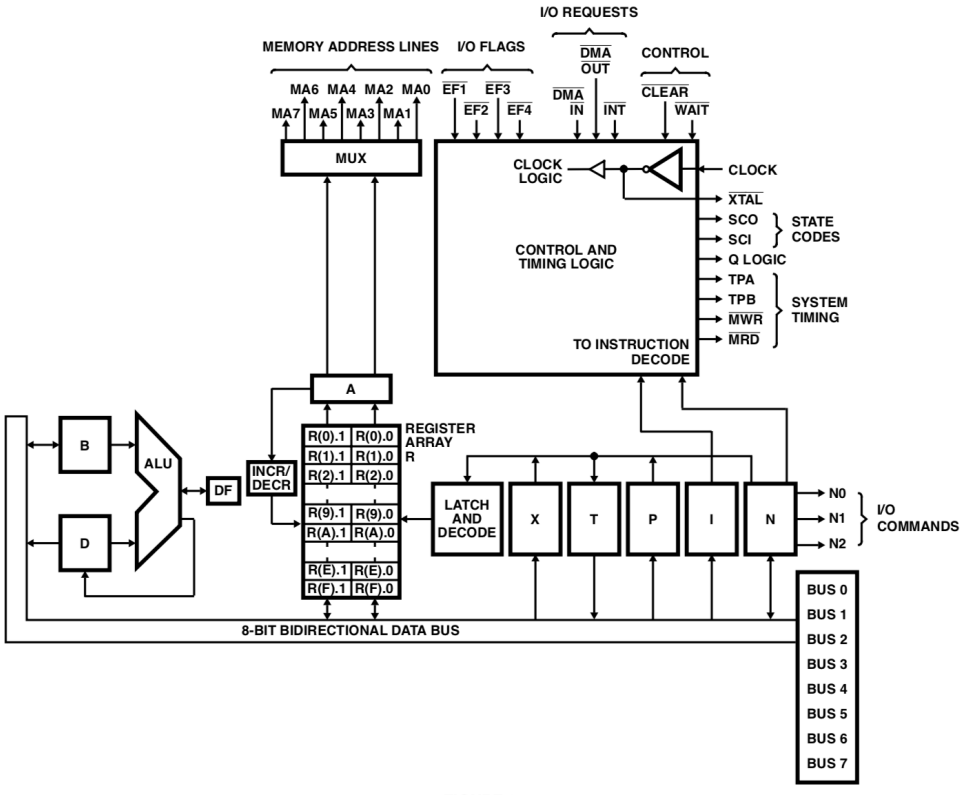
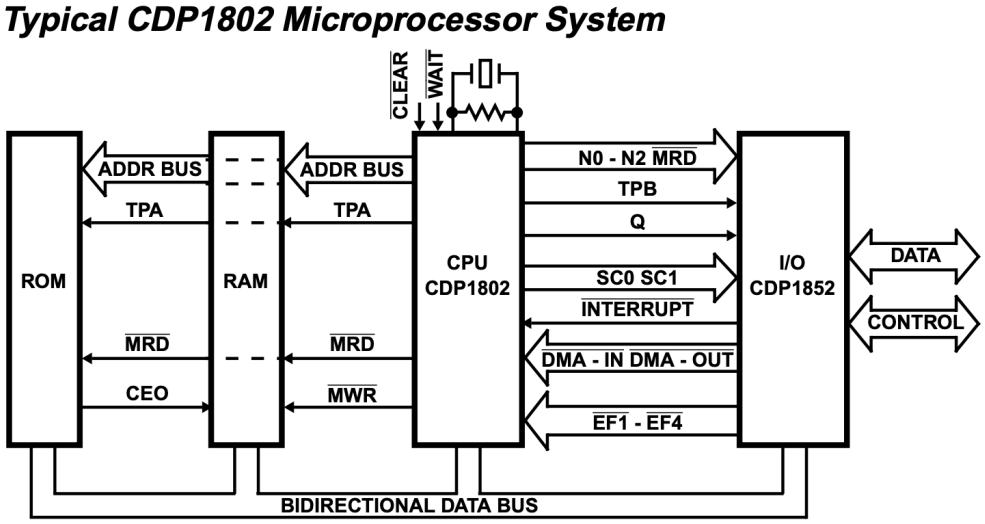


FIGURE 2

4 Memory Overview

We will implement a 124x8 bit (184K) ROM and 32x8 bit (28K) RAM. The specifications for these components which we will follow. The 1802 specification has 8 memory address lines (MA0-MA7), and uses TPA signal to latch the higher-order byte of a 16-bit memory address. The lower-order byte appears after the TPA ends.

We will also implement a byte-wide input/output port according to the CDP1852 specification. The I/O will be used to strobe data in and out from peripheral devices. The 1802 selects by turning the MRD (memory read) line high; the I/O port places the data from the peripheral device onto the data bus. The 1802 also addresses the memory so the data is read from the data bus into memory.



Although Chip-8 assumes a 4K virtual memory layout, we will be using a Chip-8 interpreter written for Cosmac Elf which will present the illusion of 4K of memory. The interpreter will be stored at the first 512 bytes of RAM. Chip-8 programs begin at address 0x200 (512).

5 Peripherals

5.1 Graphics

The CDP1861 Video Display Controller (also known as Pixie graphics) generates composite vertical and horizontal sync and allows for programmable vertical resolution for up to 64 x 128 segments. The 1861 uses the `INTERRUPT` input and I/O command lines to perform handshaking with the 1802 to set up DMA transfers (discussed in section 6). Chip-8 only had 64x32-pixel monochrome display, so it will not use the full vertical resolution. However, Super Chip-8 added 128x64-pixel mode, so if we are able to run Super Chip-8, that would be cool too!

The Video Display Controller has a framebuffer which will generate NTSC-rate video output to the VGA monitor.

5.2 Sound

A one-bit output from the microprocessor, the Q line, driven by software produces sounds through an attached speaker. Although the original chip's design is to have a single tone, we will look to customize the tone.

6 ISA

6.1 1802 ISA

Below is a summary of 1802's ISA and we intend to be faithful to the original design. The opcodes are two bytes labeled 'I' and 'N'. For this section, we refer to the 16 2-byte registers as R(X) for X the index of the register. And, we refer to 5 special purpose registers N,P,X,I, and D. Where IP is the opcode and D generally is used for data and X used as an index. R(P) is the current program counter. All instructions are 2 cycles except for CX instructions.

1802 INSTRUCTION MATRIX

		'N'															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
'I'	0	IDL	"LDN" LOAD 'D' FROM REGISTER R(N) – EXCEPT R0														
	1	"INC" INCREMENT REGISTER R(N)															
	2	"DEC" DECREMENT REGISTER R(N)															
	3	BR	BRANCH ON Q , Z , F			BRANCH ON EF1 ~ EF4 = 1				SKP	BRANCH NOT ON Q , Z , F			BRANCH ON EF1 ~ EF4 = 0			
	4	"LDA" LOAD 'D' FROM ADDRESS IN REGISTER R(N) & ADVANCE															
	5	"STR" STORE 'D' INTO ADDRESS POINTED TO BY REGISTER R(N)															
	6	IRX	OUTPUT							-	INPUT						
	7	CONTROL & MEMORY REF.				ARITHMETIC w/ CARRY				CONTROL		RST/SET 'Q'		ARITHMETIC, IMMED. w/ CARRY			
	8	"GLO" GET LOW BYTE OF REGISTER R(N), PUT IN 'D'															
	9	"GHI" GET HIGH BYTE OF REGISTER R(N), PUT IN 'D'															
	A	"PLO" PUT 'D' INTO LOW BYTE OF REGISTER R(N)															
	B	"PHI" PUT 'D' INTO HIGH BYTE OF REGISTER R(N)															
	C	LONG BRANCHES				NOP	LONG SKIPS				LONG BRANCHES			LONG SKIPS			
	D	"SEP" SET REGISTER 'P' FROM LOWER BYTE OF INSTRUCTION (N)															
	E	"SEX" SET REGISTER 'X' FROM LOWER BYTE OF INSTRUCTION (N)															
	F	LOGIC				ARITHMETIC				LOGIC & ARITHMETIC IMMEDIATE				ARITHMETIC IMMED.			

6.2 00:IDL

Wait for DMA or interrupt

6.3 0N: (except N=0) Load

Load data from the address stored in register N, R(N) to the D register.

6.4 1N: INC

Increment value of register N, R(N) .

6.5 2N: DEC

Decrements value of register N, R(N).

6.6 30: BR

Set R(P) to memory value stored at R(P).

6.7 31: BQ

Branch if Q = 1.

6.8 32: BZ

If register D equal 0 branch to memory value at R(P) otherwise advance R(P) by one.

6.9 33: BDF

Branch if DF = 1

6.10 34: B1

Branch if external flag 1 equal 1

6.11 35: B2

Branch if external flag 2 equal 1

6.12 36: B3

Branch if external flag 3 equal 1 37: B4 Branch if external flag 4 equal 1

6.13 38: SKP

Advance R(P) by 1.

6.14 39: BNQ

Branch if Q = 0

6.15 3A: BNZ

If register D not equal 0 branch to memory value at R(P) otherwise advance R(P) by one.

6.16 3B: BNF

Branch if DF not equal 1

6.17 3C: BN1

Branch if external flag 1 is 0

6.18 3D: BN2

Branch if external flag 2 is 0

6.19 3E: BN3

Branch if external flag 3 is 0

6.20 3F: BN4

Branch if external flag 4 is 0

6.21 4N: LDA

Load data from the address stored in R(N) (2 cycles) to the D register and advances the address in register N.

6.22 5N: STR

Store data in D register to address in register N (2 cycles).

6.23 60: IRX

For whatever index, x, stored in register X, increment R(X) by 1.

6.24 61-67

Write memory value at R(X) to bus and increment R(X).

6.25 68

undefined behavior

6.26 69-6F

Write bus value in R(X) and also in D register.

6.27 70

Return from a function. Set X,P to value stored at R(X) advance R(X) and set IE to 1.

6.28 71

Return and disable interrupts. IE = 0.

6.29 72: LDXA

Load to D register address stored in R(X) and advance value of R(X).

6.30 73: STXD

Store value in D register into memory at address stored in R(X) and decrements value of R(X).

6.31 74: ADC

Add memory at address in R(P), D register, and carry bit (DF). If there is a carry then carry bit (DF) is set to 1 and R(P) is advanced.

6.32 75: SDB

Subtract D register value and 1 if DF = 0 from memory stored in address held in register R(X) and store back in D register..

6.33 76:SHRC

Shift right with carry. Shift right but the most significant bit is now the carry bit (DF).

6.34 77: SMB

Subtract memory stored in address held in register R(X) and 1 if DF = 0 from D register value and store back in D register.

6.35 78: SAV

Save register T to address in R(X)

6.36 79: MARK

Save XP to T register and XP to address in R(2) then set X to P and decrement R(2)

6.37 7A: REQ

Reset Q to 0

6.38 7B: SEQ

Set Q to 1

6.39 7D: SDBI

Subtract D register value and 1 if DF = 0 from memory stored in address held in register R(P) and increment R(P)'s value.

6.40 7E:SHLC

Shift left with carry. Shift left but the least significant bit is now the carry bit (DF).

6.41 7F: SMBI

Subtract memory stored in address held in register R(P) and 1 if DF = 0 from D register value and increment R(P)'s value.

6.42 8N: GLO

Get lower (GLO) byte R(N) and store it in D register.

6.43 9N: GHI

Get higher (GHI) byte of R(N) and store it in D register.

6.44 AN: PLO

Write to lower byte of R(N) with value of D register.

6.45 BN: PHI

Write to higher byte of R(N) with value of D register.

6.46 C0:LBR

Long branch (ie set R(P) upper byte to memory at R(P) and R(P) lower byte to memory R(P+1)

6.47 C1: LBQ

Long barnch if Q = 1

6.48 C2: LBZ

If D = 0, long branch

6.49 C3: LBDF

If DF = 1, long branch

6.50 C4: NOP

continue

6.51 C5: LNQ

Long skip if Q=0

6.52 C6: LSNZ

if D not 0, long skip

6.53 C7: LSNF

Long skip if DF = 0

6.54 C8: NLBR

Long skip R(P) +=2.

6.55 C9: LBNQ

Long branch if Q = 0

6.56 CA: LBNZ

If D not 0, long branch

6.57 CB: LBNF

If DF = 0, long branch

6.58 CC: LSIE

Long skip if IE =1 (interrupt enabled)

6.59 CD: LSQ

Long skip if Q = 1

6.60 CE: LSZ

Long Skip if D = 0

6.61 CF: LSDF

Long skip if DF = 1

6.62 DN: SEP

Set P to N.

6.63 EN: SEX

Set X to N.

6.64 F0

For value in X register, load from address in R(X) to D register.

6.65 F1: OR

Computes the OR of value of D register and memory in address held in R(X).

6.66 F2: AND

Computes the AND of value of D register and memory in address held in R(X).

6.67 F3: XOR

Computes the XOR of value of D register and memory in address held in R(X).

6.68 F4: XOR

Computes ADD of value of D register and memory in address held in R(X).

6.69 F5: SD

Computes memory value in address held in R(X) minus D register's value. Afterward DF = 0 if carrying was needed.

6.70 F6: SHR

Bit shift D register to the right and hold lowest order bit in carry bit (DF).

6.71 F7: SM

Subtract memory value in address held in R(X) from D and store in D.

6.72 F8: LDI

For index held in P register, load from address in R(P) into D register. Then increment the value of R(P).

6.73 F9: ORI

Computes the OR of value of D register and memory held in address held in R(P) and also increments value in P register.

6.74 FA: ANI

Computes the AND of value of D register and memory held in address held in R(P) and also increments value in P register.

6.75 FB: XRI

Computes the XOR of value of D register and memory held in address held in R(P) and also increments value in P register.

6.76 FC: ADI

Computes ADD of value of D register and memory held in address held in R(P) and also increments value in P register.

6.77 FD: SDI

Computes memory value in address held in R(P) minus D register's value and advance address in R(P). Afterward DF = 0 if carrying was needed.

6.78 FE: SHL

Shift left and place most significant bit in carry bit (DF).

6.79 FF: SMI

Subtract memory value in address held in R(P) from D and store in D and increment address in R(P).

6.80 INTERRUPT

save x and p into register T (XP), set P to 1 and X to 2 and IE to 0

6.81 DMA-IN

Read from bus to memory addressed by R(0) then advance R(0).

6.82 DMA-OUT

Write to buss from memory addressed by R(0) and advance R(0).

7 Interrupts and DMA

The interrupt, DMA in, and DMA out inputs are sampled by the CPU. Interrupts can have higher priority over regular I/O by internal flags in the I/O controller. When an interrupt is serviced, the register r(1) is used as the PC. The interrupt action requires one machine cycle.

DMA The state transition diagram from the 1802 specification shows the priority of DMA and interrupts.

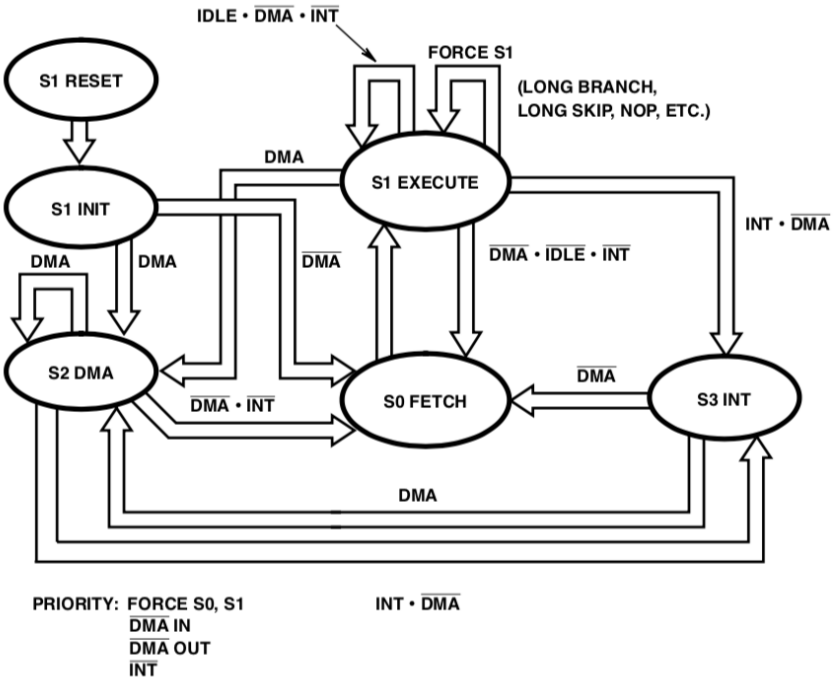


FIGURE 25. STATE TRANSITION DIAGRAM

8 Testing using Verilator

We will compile 1802 object code from C programs using the LLC 1802 compiler (link), or simply program the hex values by hand if that turns out to be easier.

We can compare the processor behavior (register state, PC, program output) with an 1802 Instruction-level simulator. Either Emma02 (link) or TinyElf (link) could work for this purpose.