Emulators and Web Assembly

Dr. Charles R. Severance

online.dr-chuck.com

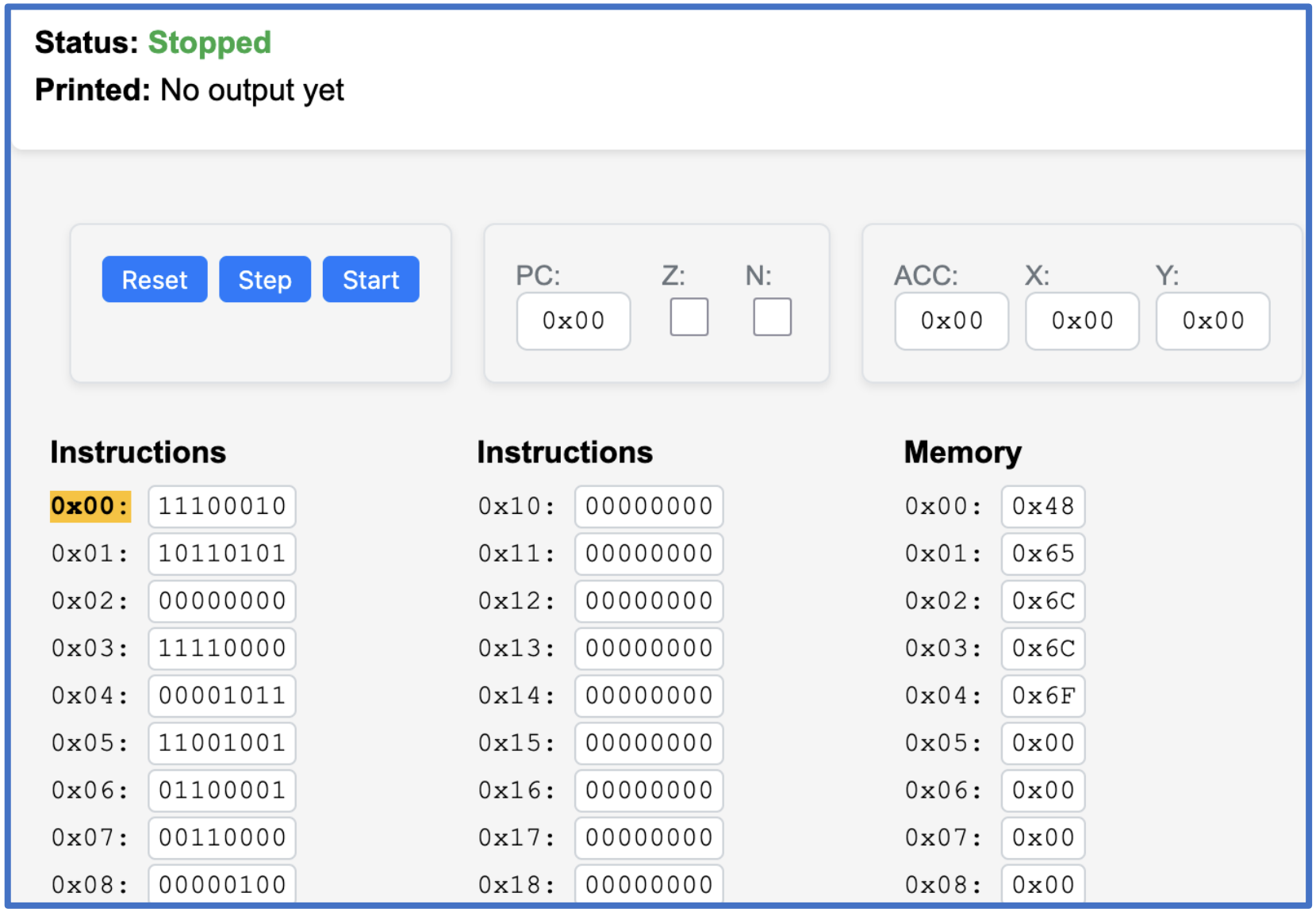
Outline

- Start from our "two instruction CPU" from the previous section
- Hexadecimal Notation
- Review: What is Programming from PY4E
- Real CPUs – Intel x86, ARM, and MOS 6502
- The CDC6504 – Mostly 6502 with homage to CDC 6500
  - Architecture
  - Machine Language
  - Assembly Language

Uppercase a String

```
CLX
loop:
LDA $00,X
BEQ done
CMP #'a'
BMI cont
SBC #0x20
STA $00,X
cont:
INX
JMP loop
done:
BRK

DATA 'Hello'
```

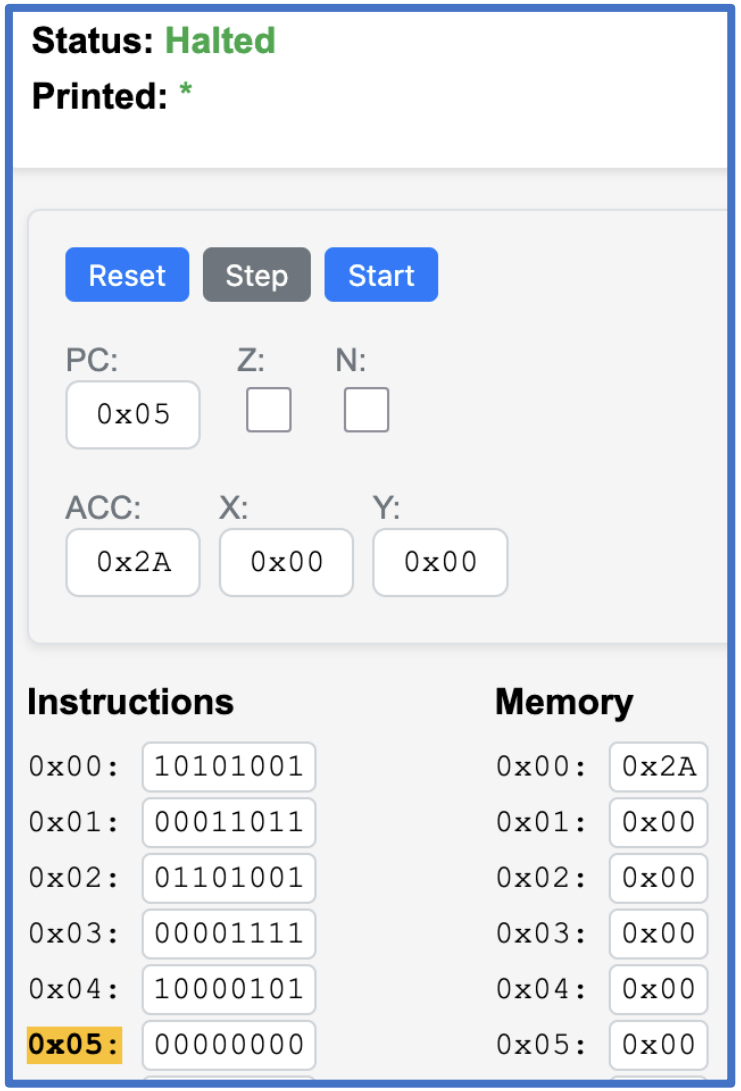


https://www.ca4e.com/tools/cdc6504/cpu-emulator.js

```
 executeStep() {
    // Read instruction from instruction memory
    const instruction = this.cpu.instructions[this.cpu.pc];

    // Decode 6502 opcodes
    if (instruction === 0x00) { // BRK - Break/Halt
        this.printString(); // Print data memory on BRK
        this.running = false;
    }
...
    else {
        console.log(`ERROR: unknown instruction - halting`);
        this.running = false;
    }

    // Update PC (if not already updated by branch/jump)
    if (pcIncrement > 0) {
        this.cpu.pc = (this.cpu.pc + pcIncrement) & 0xFF;
    }
}
```



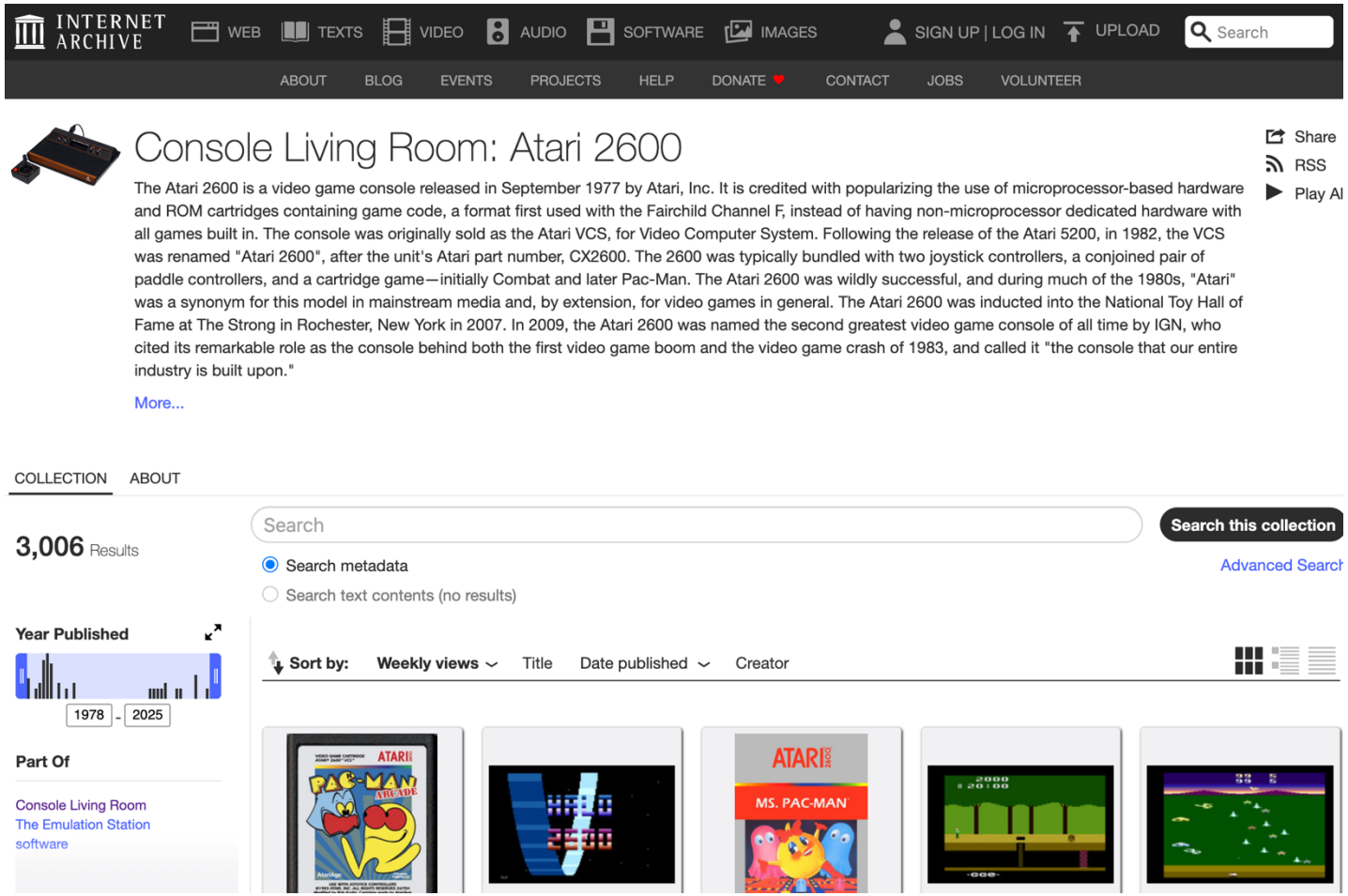
https://www.ca4e.com/tools/cdc6504/cpu-emulator.js

```
else if (instruction === 0xA9) { // LDA # - Load accumulator immediate
    const immediate = this.cpu.instructions[this.cpu.pc + 1];
    this.cpu.acc = immediate;
    this.updateCompareFlags(this.cpu.x, immediate);
    pcIncrement = 2;
} else if (instruction === 0xE9) { // SBC # - Subtract immediate
    const immediate = this.cpu.instructions[this.cpu.pc + 1];
    const diff = this.cpu.acc - immediate;
    this.cpu.acc = diff & 0xFF;
    this.updateStatusFlags(this.cpu.acc);
    pcIncrement = 2;
 } else if (instruction === 0xB5) { // LDA $nn,X - Load indexed by X
    const baseAddr = this.cpu.instructions[this.cpu.pc + 1];
    const xValue = Number(this.cpu.x);
    const effAddr = (baseAddr + xValue) & 0xFF;
    this.cpu.acc = this.cpu.memory[effAddr];
    this.updateStatusFlags(this.cpu.acc);
    pcIncrement = 2;
}
```

Building Emulators for Historical CPUs

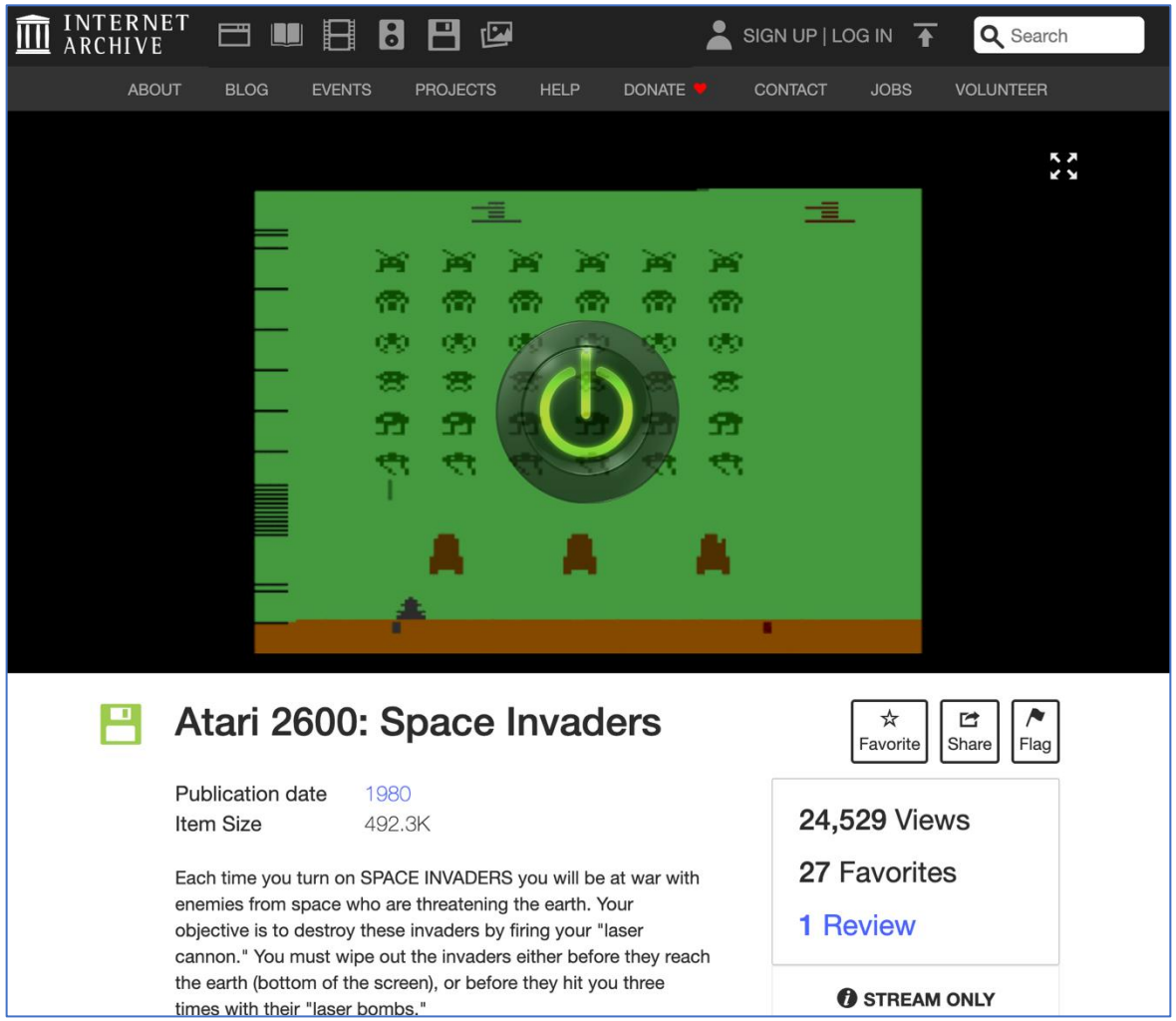
- It is not all that difficult to build an emulator for an older CPU (emulator.js is 1300 lines of code – 6502 is 4528 transistors)
- Modern processors on our phones and computers are so fast that JavaScript in a browser can emulate historical machine code faster than the original hardware
- The computer / phone / browser graphics are also far better
- And it can be great fun

Archive.org: Internet Arcade

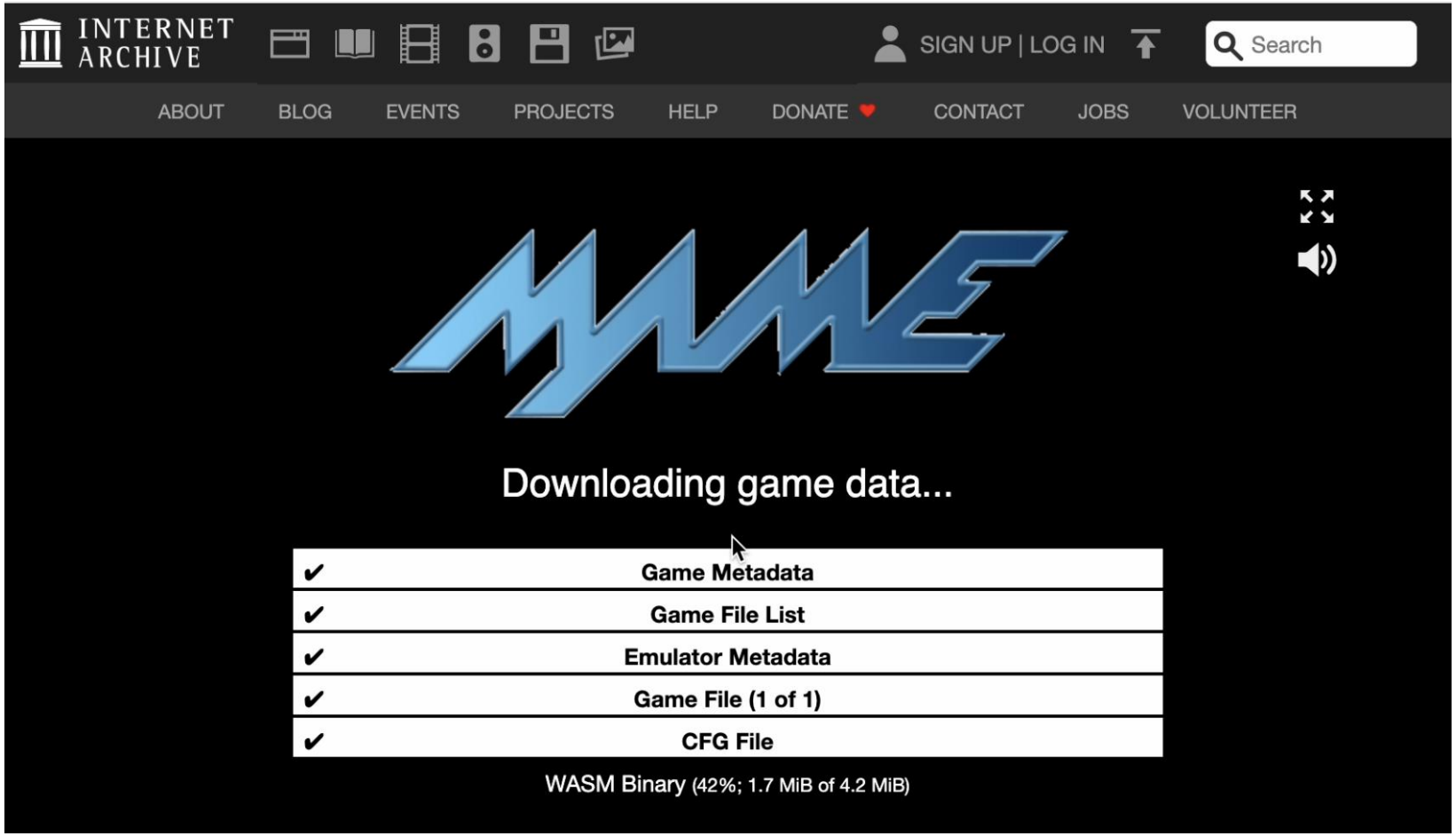


Atari 2600
Space Invaders

The Atari 2600 has a 6502 processor.



Space Invaders Startup – JSMAME Emulator



WASM – Web Assembly

- There is a machine language emulator built into browsers called Web Assembly
- For a wide range of software, compiling to WASM and running in JavaScript is enough
- WASM is a different syntax than most Assembly languages – but it still compiles to a binary-machine code
- C Programming for Everybody (www.cc4e.com) C Playground compiles C to WASM and then runs the WASM in the user's browser

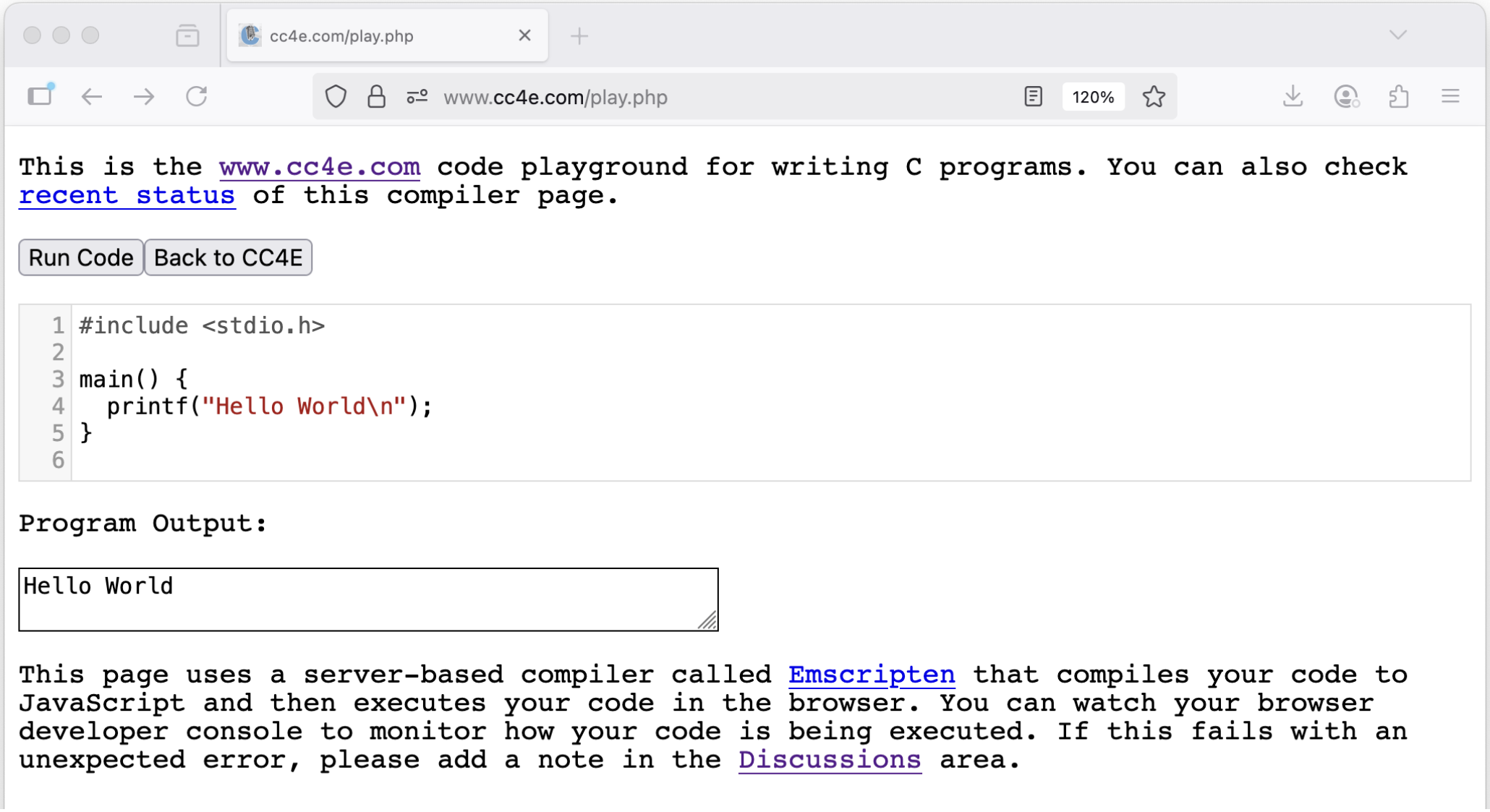
WASM versus traditional Assembly

- **Sandboxed by design**: Wasm runs in a strict sandbox with no direct access to memory, files, or OS — unlike native assembly which runs with full process privileges.
- **Safe execution model**: Structured control flow, validated code, and no undefined jumps help prevent many classic crashes and exploits.
- **Portable bytecode**: Same Wasm runs on any browser or runtime; traditional assembly is tied to a specific CPU (x86, ARM, etc.).
- **Abstract machine**: Wasm targets a virtual stack machine, not real hardware — traditional assembly maps directly to CPU instructions.
- **Optimized by the host**: Browsers and runtimes validate and compile Just In Time (JIT) - compile Wasm safely, while native assembly relies on machine code produced elsewhere with no validation.

```
;; Hello World WASM Example
(module
  (import "console" "log" (func $log (param i32 i32)))
  (memory 1)
  (data (i32.const 0) "Hello, World!")
  (func $main (result i32)
    (call $log (i32.const 0) (i32.const 13))
    (i32.const 42)
  )
  (export "main" (func $main))
)
```

WASM Hello World

```
00000000: 00 61 73 6d 01 00 00 00 01 0a 02 60 02 7f 7f 00 |.asm.......`....|
00000010: 60 00 01 7f 02 0f 01 07 63 6f 6e 73 6f 6c 65 03 |`.......console.|
00000020: 6c 6f 67 00 00 03 02 01 01 05 03 01 00 01 07 11 |log.............|
00000030: 02 06 6d 65 6d 6f 72 79 02 00 04 6d 61 69 6e 00 |..memory...main.|
00000040: 01 0a 0c 01 0a 00 41 00 41 0d 10 00 41 2a 0b 0b |......A.A...A*..|
00000050: 13 01 00 41 00 0b 0d 48 65 6c 6c 6f 2c 20 57 6f |...A...Hello, Wo|
00000060: 72 6c 64 21                                     |rld!|
```


cc4e.com/play.php
www.cc4e.com/play.php
120%
This is the www.cc4e.com code playground for writing C programs. You can also check recent status of this compiler page.
Run Code
Back to CC4E
1 #include <stdio.h>
2
3 main() {
4 printf("Hello World\n");
5 }
6
Program Output:
Hello World
This page uses a server-based compiler called Emscripten that compiles your code to JavaScript and then executes your code in the browser. You can watch your browser developer console to monitor how your code is being executed. If this fails with an unexpected error, please add a note in the Discussions area.

From the 6502 to the Present

A lot has changed since the 6502 was available as a $25 microprocessor

Modern CPUs

- Since the 1970's there have been about 50 distinct CPU families
  - Lots of research and innovation
  - Adoption and switching
  - Generational turn over
    - From CISC to RISC
    - From mainframe to microprocessor
- Today there are two major CPU families in use (computers + phones)
  - ARM based processors (90%)
  - Intel x86 processors (10%)

Case Study: Apple Processors

- MOS 6502 – Apple I / Apple II – 1977 - 1993
- Motorola MC68000 – Macintosh 1984-1994
- ARM – Apple Newton - 1993
- IBM PowerPC – Macintosh 1994 – 2006
- Intel x86 – Macintosh (2006-2020)
- Samsung ARM – iPhone (2007-2009)
- Apple ARM – iPhone (2010-Present)
- Apple ARM – Macintosh (2020-Present)

- Apple currently redesigns their CPUs based on workload over time – Image Processing / Audio / Video / Gaming / Artificial Intelligence

Fun Aside – Apple Built and Shipped Machine Code Translators

- 2005 Moved from PowerPC to Intel Processors – Rosetta 1
- 2020 Moved from Intel to Apple ARM Processors – Rosetta 2

- Translated machine code as the application was launched
- Greatly eases what would otherwise be a major upheaval for customers

https://en.wikipedia.org/wiki/Sophie_Wilson

Fun Aside – the 6502 influenced ARM

- Acorn's 6502-based BBC Micro
- The ARM designers were 6502 experts
- The BBC Micro was used to build and simulate the ARM design
- ARM1 had ~25,000 transistors
- Currently ARM based processors are the most widely used processors in the world.

- AI: Describe the historical connection between the 6502, ARM, and Raspberry Pi

Sophie Wilson

Summary

- CPU Evolution over time

Acknowledgements / Contributions

These slides are Copyright 2025- Charles R. Severance (online.dr-chuck.com) as part of www.ca4e.com and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here