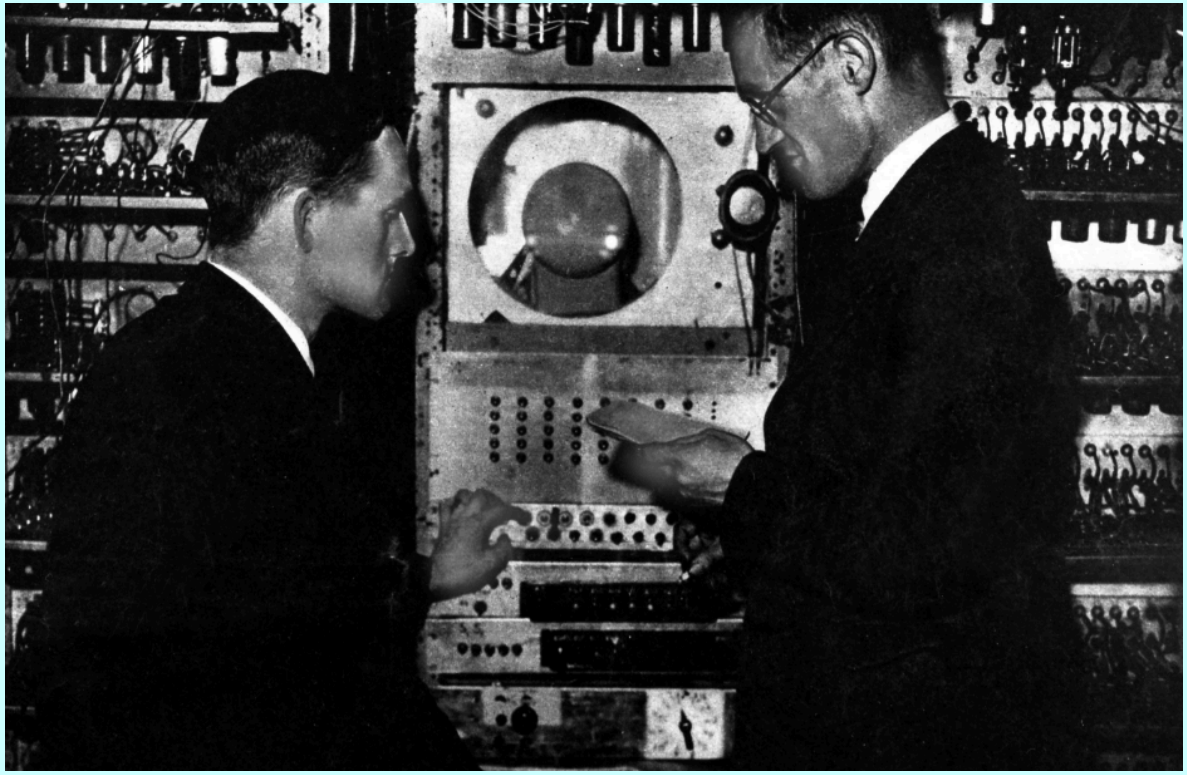
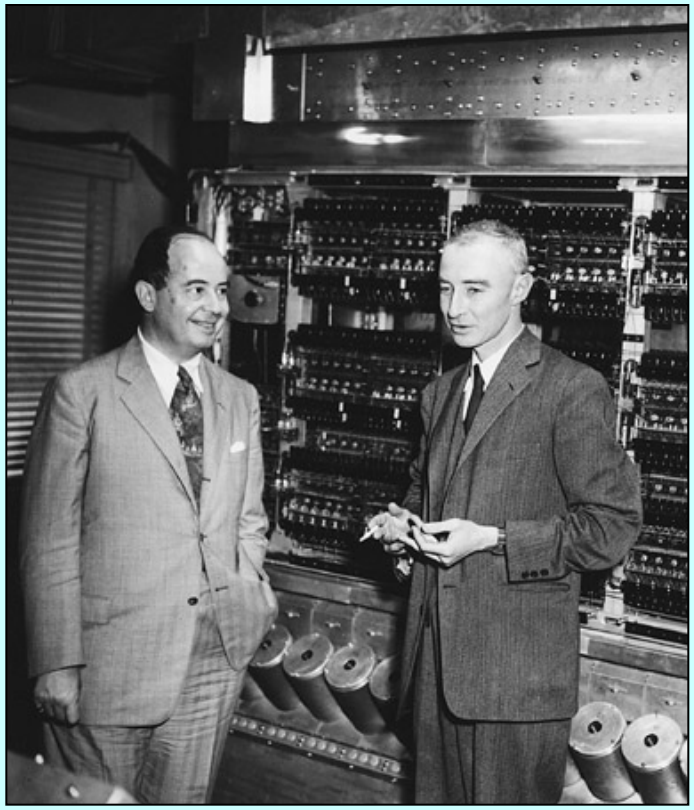
Stored Program Machines



Chris Gregg, Based on slides by Eric Roberts
CS 208E
October 11, 2021

The von Neumann Architecture

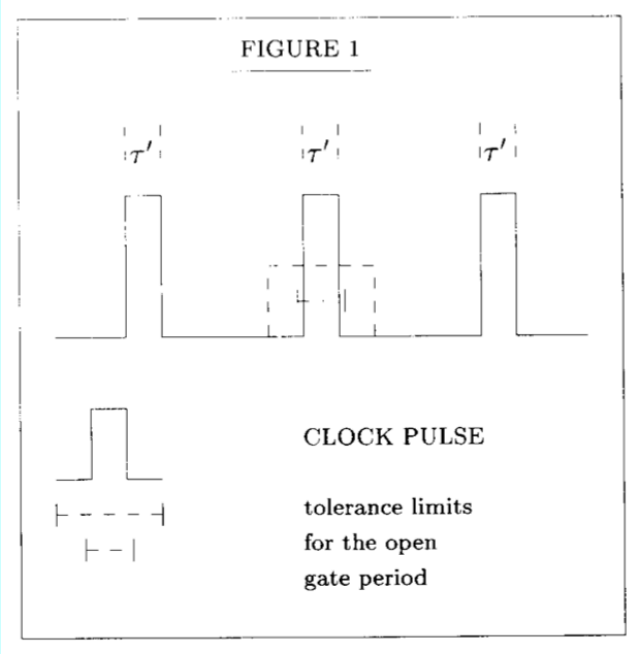
- One of the foundational ideas of modern computing—traditionally attributed to John von Neumann although others can make valid claims to the idea—is that code is stored in the same memory as data. This concept is called the ***stored programming model***.
- The next few slides introduce the Manchester Baby, which was the first stored-program computer. In the rest of today's class, we will describe the operation of a slightly more powerful machine that Eric Roberts nicknamed "Toddler".



John von Neumann and J. Robert Oppenheimer

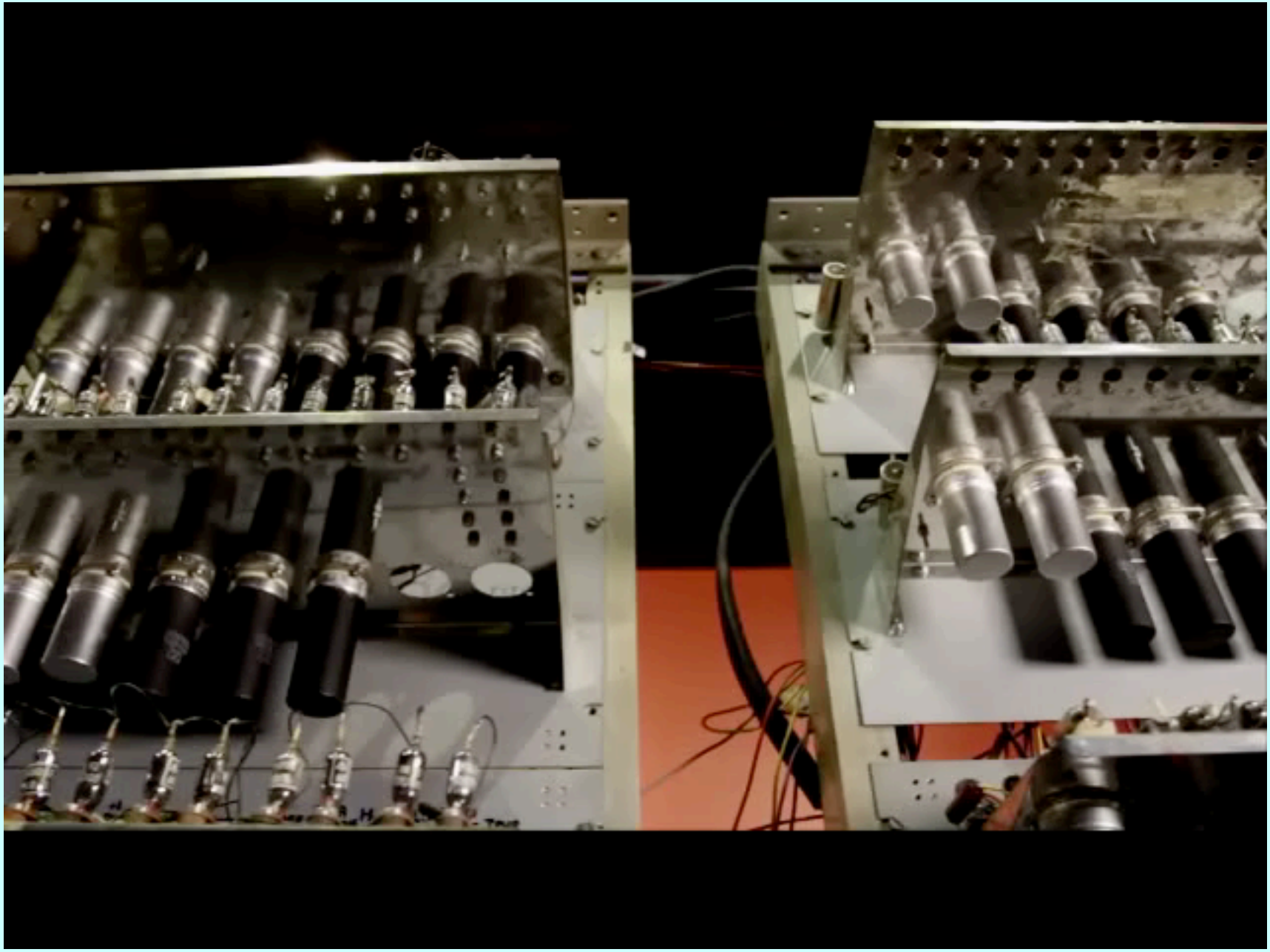
The von Neumann Architecture

- Links:
  - First Draft of a Report on the EDVAC:
    - https://en.wikipedia.org/wiki/First_Draft_of_a_Report_on_the_EDVAC
  - Von Neumann Architecture - Computerphile:
    - https://www.youtube.com/watch?v=Ml3-kVYLNr8



A figure from the First Draft of a Report on the EDVAC

The Manchester Baby



Structure of the Toddler Machine

	0*x*	1*x*	2*x*	3*x*	4*x*	5*x*	6*x*	7*x*	8*x*	9*x*
0		+000	+000	+000	+000	+000	+000	+000	+000	+000
1	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
2	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
3	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
4	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
5	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
6	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
7	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
8	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
9	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000

AC

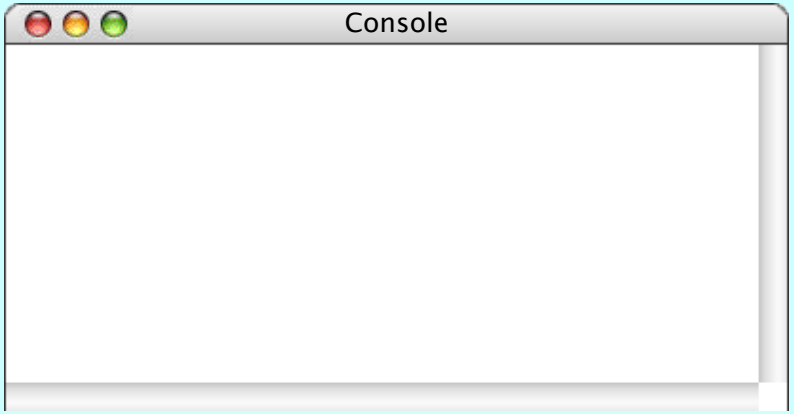
+	0	0	0

PC

0	0

IR

+	0	0	0



The Toddler Instruction Set

1*xx*	**LOAD** *xx*	Loads the value from address *xx* into the AC
2*xx*	**STORE** *xx*	Stores the value from AC into address *xx*
3*xx*	**ADD** *xx*	Adds the value at address *xx* to the AC
4*xx*	**SUB** *xx*	Subtracts the value at address *xx* from AC
500	**HALT**	Halts the machine

8*xx*	**INPUT** *xx*	Reads a value into address *xx*
9*xx*	**OUTPUT** *xx*	Prints the value in address *xx*

The Add-Two-Numbers Program

```
(01) +850      INPUT 50
(02) +851      INPUT 51
(03) +150      LOAD 50
(04) +351      ADD 51
(05) +252      STORE 52
(06) +952      OUTPUT 52
(07) +500      HALT
```

The Instruction Cycle

1. *Fetch the current instruction.* In this phase, Toddler finds the word from the memory address specified by the **PC** and copies its value into the IR.

2. *Increment the program counter.* Once the current instruction has been copied into the IR, Toddler adds one to the PC so that it points to the next instruction.

3. *Decode the instruction in the instruction register.* The value copied into the IR is a three-digit integer. To use it as an instruction, Toddler must divide the instruction word into its *opcode* and *address* components.

4. *Execute the instruction.* Once the operation code and address field have been identified, the Toddler processor must carry out the steps necessary to perform the indicated action.

The Add-Two-Numbers Program

	0*x*	1*x*	2*x*	3*x*	4*x*	5*x*	6*x*	7*x*	8*x*	9*x*
0		+000	+000	+000	+000	+000	+000	+000	+000	+000
1	+850	+000	+000	+000	+000	+000	+000	+000	+000	+000
2	+851	+000	+000	+000	+000	+000	+000	+000	+000	+000
3	+150	+000	+000	+000	+000	+000	+000	+000	+000	+000
4	+351	+000	+000	+000	+000	+000	+000	+000	+000	+000
5	+252	+000	+000	+000	+000	+000	+000	+000	+000	+000
6	+952	+000	+000	+000	+000	+000	+000	+000	+000	+000
7	+500	+000	+000	+000	+000	+000	+000	+000	+000	+000
8	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000
9	+000	+000	+000	+000	+000	+000	+000	+000	+000	+000

AC

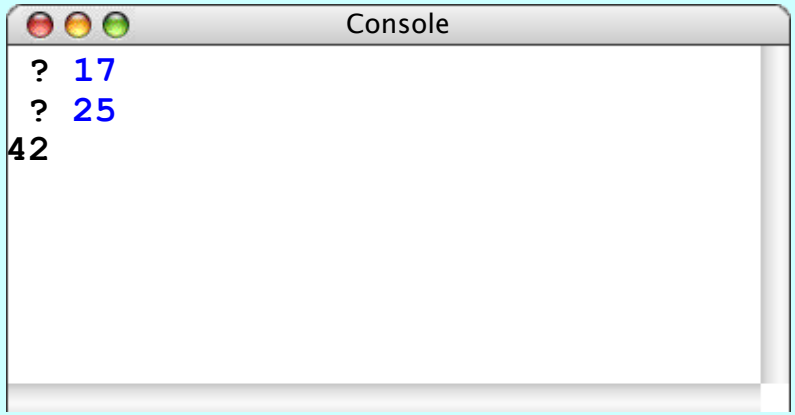
+0[illegible]

PC

0[illegible]

IR

+[illegible]



The Toddler Instruction Set

1*xx*	**LOAD** *xx*	Loads the value from address *xx* into the AC
2*xx*	**STORE** *xx*	Stores the value from AC into address *xx*
3*xx*	**ADD** *xx*	Adds the value at address *xx* to the AC
4*xx*	**SUB** *xx*	Subtracts the value at address *xx* from AC
500	**HALT**	Halts the machine
5*xx*	**JUMP** *xx*	Takes the next instruction from address *xx*
6*xx*	**JUMPZ** *xx*	Jumps to *xx* if the AC is zero
7*xx*	**JUMPN** *xx*	Jumps to *xx* if the AC is negative
8*xx*	**INPUT** *xx*	Reads a value into address *xx*
9*xx*	**OUTPUT** *xx*	Prints the value in address *xx*

The Countdown Program

	assembly language
(01) +111	`start:  LOAD ten`
(02) +212	`        STORE i`
(03) +709	`loop:   JUMPN done`
(04) +912	`        OUTPUT i`
(05) +112	`        LOAD i`
(06) +410	`        SUB one`
(07) +212	`        STORE i`
(08) +503	`        JUMP loop`
(09) +500	`done:   HALT`
(10) +001	`one:    1`
(11) +010	`ten:    10`
(12) +000	`i:      0`

Representing Constants

- Just as was true for the Analytical Engine, constants in the Toddler machine need to be stored in one of the memory addresses, as illustrated by the following lines from the assembly language version of **`Countdown.td`**:

```
one:    1
ten:    10
```

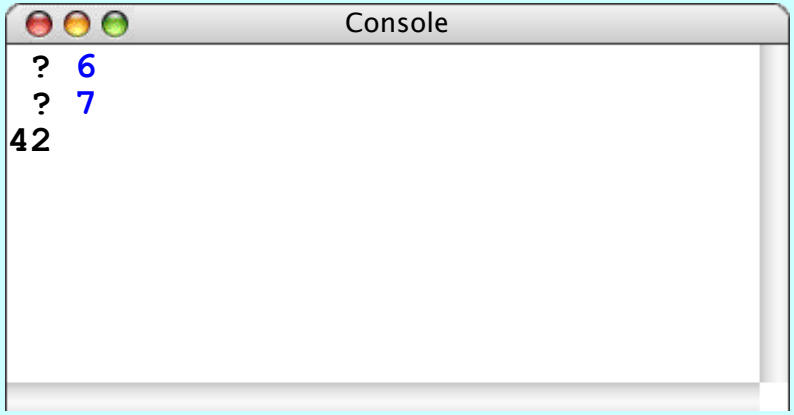
- The instruction **`LOAD ten`** then refers to a memory address that contains the value 10.
- Toddler also allows you to write

```
LOAD #10
```

  which finds space for the constant 10 at the end of the program and then fills in the **`LOAD`** instruction with the address of that constant.

Exercise: Multiply Two Numbers

- How would you write a Toddler program to multiply two nonnegative numbers, even though the machine has no multiply instruction?



The End