Eric Roberts
MLA 321

Handout #7
January 26, 2016

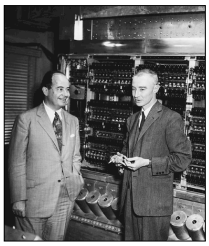
Stored-Program Machines

Stored Program Machines

Eric Roberts
MLA 321
January 26, 2016

The von Neumann Architecture

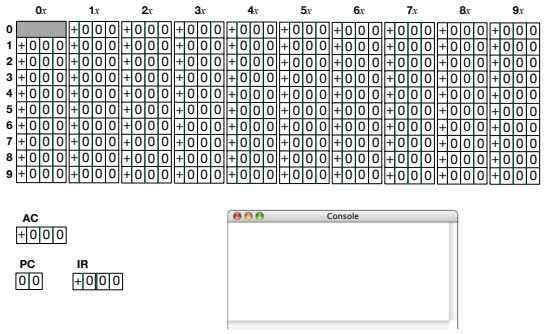
- One of the foundational ideas of modern computing—traditionally attributed to John von Neumann although others can make valid claims to the idea—is that code is stored in the same memory as data. This concept is called the ***stored programming model***.
- The next few slides introduce the Manchester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



John von Neumann and J. Robert Oppenheimer

The Manchester Baby

Structure of the Toddler Machine



The Toddler Instruction Set

1*xx*	**LOAD** *xx*	Loads the value from address *xx* into **AC**
2*xx*	**STORE** *xx*	Stores the value from **AC** into address *xx*
3*xx*	**ADD** *xx*	Adds the value at address *xx* to **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 **AC** is zero
7*xx*	**JUMPN** *xx*	Jumps to *xx* if **AC** is negative
8*xx*	**INPUT** *xx*	Reads a value into address *xx*
9*xx*	**OUTPUT** *xx*	Prints the value in address *xx*

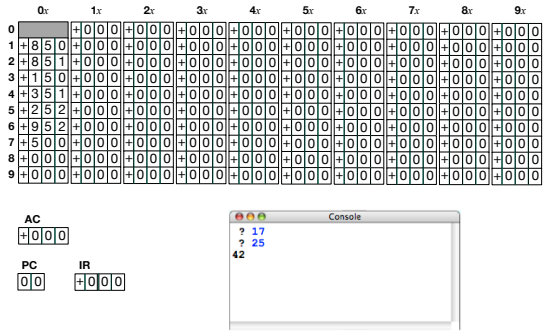
The Add-Two-Numbers Program

```
(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



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 03`
(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 Concept of a Stack

- A ***stack*** is a data structure in which the elements are accessible only in a ***last-in/first-out*** order. The operations on a stack are **`push`**, which adds a value to the top of the stack, and **`pop`**, which removes and returns the top value.
- One of the most common metaphors for the stack concept is a spring-loaded storage tray for dishes. Adding a new dish to the stack pushes any previous dishes downward. Taking the top dish away allows the dishes to pop back up.
- Stacks are important in von Neumann machines because function calls obey a last-in/first-out discipline.

The Toddler System Stack

- Like all modern hardware, the Toddler machine implements a stack in hardware to simplify dividing programs up into independent functions.
- The Toddler stack lives at the highest addresses in memory, so the bottom of a stack is at address 99, and the stack grows toward lower memory addresses.
- The address of the element at the top of the stack is stored in the register **SP**. If the **SP** is 00, that means the stack is empty.
- Pushing a value on the stack corresponds to subtracting one from the **SP** and then storing a value in the resulting address.
- Popping the top value from the stack reverses the process by taking the current contents of the word addressed by **SP** and then adding one to **SP**.

Functions and Stacks

- The **CALL** instruction pushes the current value of the **PC** (which has already been incremented to refer to the next instruction) on the stack. This value is called the ***return address***.
- The **RETURN** instruction pops the top value on the stack into the **PC**, which has the effect of returning to the point just after the **CALL** instruction.

The Extended Instruction Set

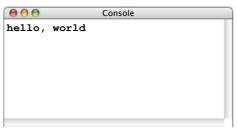
-1*xx*	**LOADX** *xx*	Loads the value from address *xx* into **XR**
-2*xx*	**STOREX** *xx*	Stores the value from **XR** into address *xx*
-3*xx*	**LOAD** *xx***(XR)**	Loads **AC** with the contents of *xx* + **XR**
-4*xx*	**STORE** *xx***(XR)**	Stores **AC** into address *xx* + **XR**
-500	**RETURN**	Returns from a function
-5*xx*	**CALL** *xx*	Call the function at address *xx*
-6*xx*	**PUSH** *xx*	Push the contents of *xx* on the stack
-7*xx*	**POP** *xx*	Pops the top element on the stack into *xx*
-8*xx*	**INCHAR** *xx*	Reads a character code into address *xx*
-9*xx*	**OUTCHAR** *xx*	Prints the character code in address *xx*

Exercise: Multiply as a Function

- Rewrite the **Multiply.td** program so that it defines a function called **mult** that takes values in the variables **n1** and **n2** and returns its answer in a variable called **result**.
- Use that function to write a program called **Factorial.td** that computes the factorial of an integer. The largest factorial that fits in three digits is 6!, so a sample run might look like this:

Class Example: Hello, World

- The **INCHAR** and **OUTCHAR** instructions are similar to **INPUT** and **OUTPUT** except that they read and write the numeric representation of a single character.
- In class, I'll go over three different implementations of a program that prints the string "hello, world" on the console.



Self-Modifying Code

- One of the defining features of the von Neumann architecture is that instructions and data are stored in the same memory. That fact makes it possible for programs to modify their own instructions by treating them just like any other numeric data.
- The **HelloWorld2.td** program uses this technique to create an instruction that prints a character from the address that is the start of the string **"hello, world"** plus the value of the index **i**. It then stores that instruction in the program and executes it.
- Programs that change their own instructions are said to be ***self-modifying***. In early machines, this strategy was often the only way to accomplish certain operations. Today, it is generally seen as a dangerous programming practice.

The Internet Worm

The New York Times

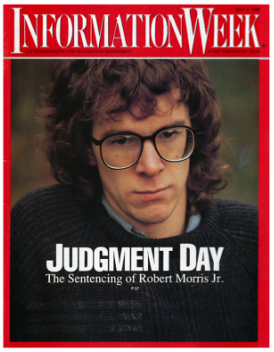
'Virus' in Military Computers Disrupts Systems Nationwide

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Registration Off Since 1984 Vote

Robert Morris Jr.



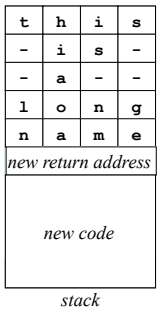
How the Morris Worm Worked

Storage for local variables in Unix is provided by a stack, which grows toward low memory addresses as functions are called.

The **`fingerd`** code allocates a stack buffer to hold the user name, which might be declared like this:

```
char buffer[20];
```

If the string supplied is too long, it will overwrite the contents of the stack and allow the worm to execute the inserted code.



Index Registers

- The **`HelloWorld3.td`** program avoids the self-modifying strategy by using the Toddler machine's ***index register*** (**`XR`**), which automatically adds the contents of the index register to the address given in a **`LOAD`** or **`STORE`** instruction.
- The **`LOADX`** and **`STOREX`** instructions load and store the contents of the **`XR`** itself. Adding the suffix **`(XR)`** to a **`LOAD`** or **`STORE`** instruction changes what memory address is used.