Extending Toddler

-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*

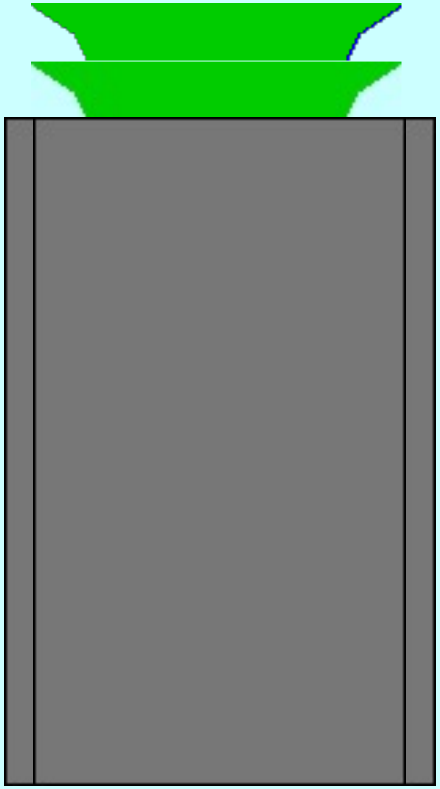
Chris Gregg, based on slides by Eric Roberts
CS 208E
October 15, 2021

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.

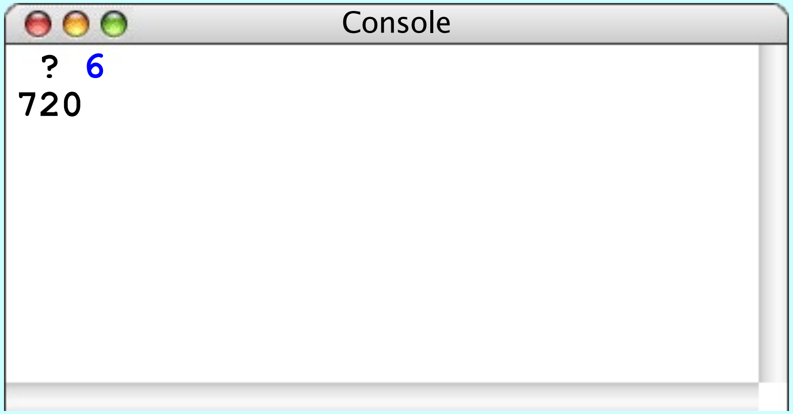
The Extended Instruction Set

-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:



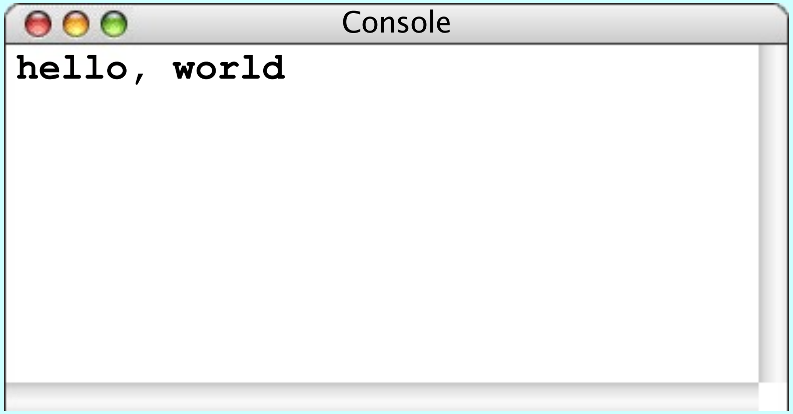
```
 ? 6
720
```

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*

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.
- The rest of this lecture develops three implementations of a program that prints the string “hello, world” on the console.



```
hello, world
```

Hello World: The Brute Force Version

```
/*
 * File: HelloWorld1.td
 * --------------------
 * Writes out "hello, world" character by character.
 */

start:  OUTCHAR #104      /* The code for the character 'h' */
        OUTCHAR #101      /* 'e' */
        OUTCHAR #108      /* 'l' */
        OUTCHAR #108      /* 'l' */
        OUTCHAR #111      /* 'o' */
        OUTCHAR #44       /* ',' */
        OUTCHAR #32       /* ' ' */
        OUTCHAR #119      /* 'w' */
        OUTCHAR #111      /* 'o' */
        OUTCHAR #114      /* 'r' */
        OUTCHAR #108      /* 'l' */
        OUTCHAR #100      /* 'd' */
        OUTCHAR #10       /* The newline character ('\n') */
        HALT
```

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.

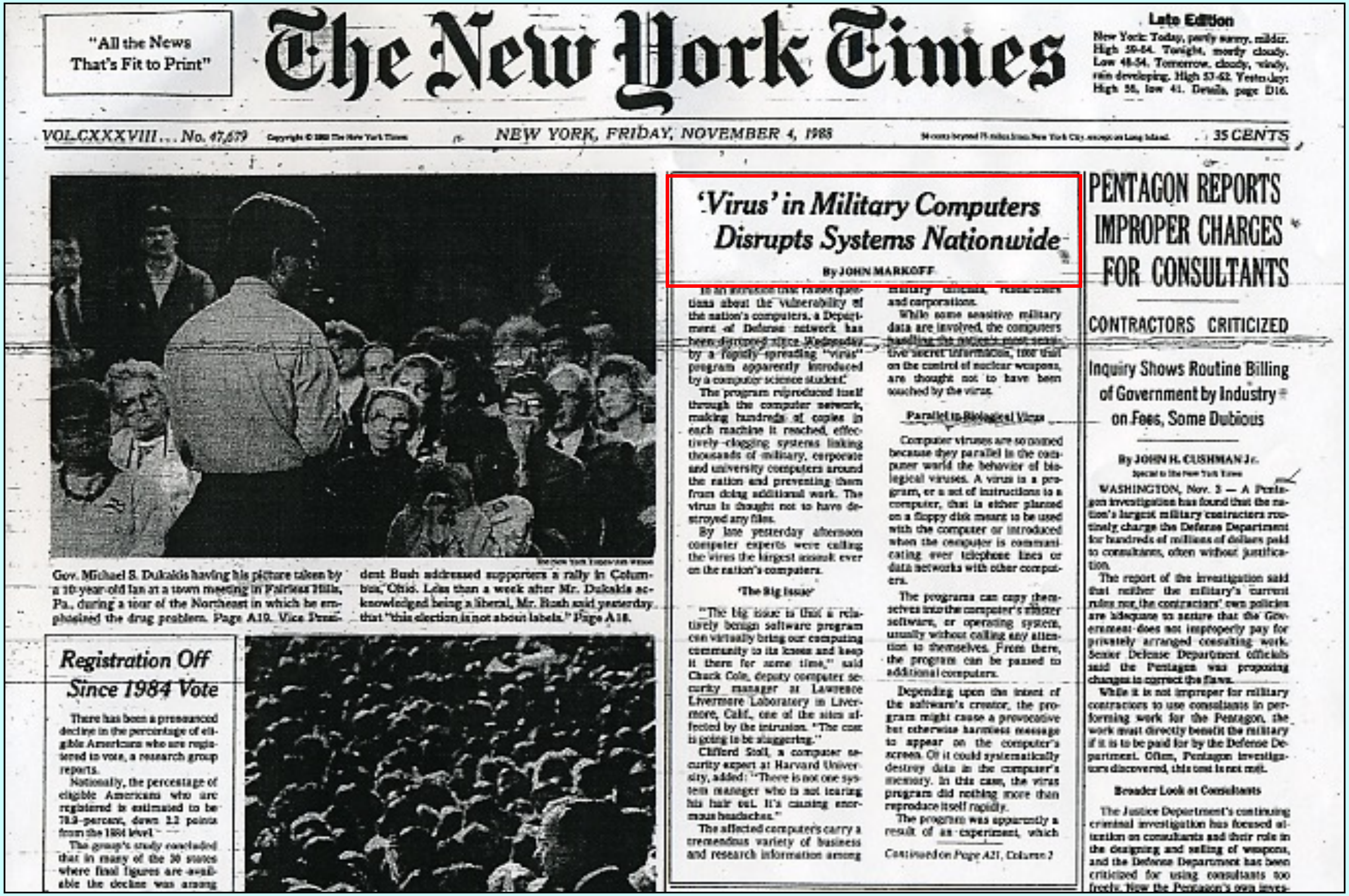
Hello World: Self-Modifying Code

```
/*
 * File: HelloWorld2.td
 * --------------------
 * Writes out "hello, world" using self-modifying code.
 */

start:  LOAD #msg         /* Load the address of the string */
        CALL strout       /* Call the function to output a string */
        HALT              /* And halt */

strout: STORE addr        /* Store current address */
        LOAD ldins        /* Load a word with a LOAD 0 instruction */
        ADD addr          /* Add the address offset */
        STORE patch       /* Store the LOAD in the next word */
patch:  0                 /* This will get filled in */
        JUMPZ ret         /* A zero character marks the end */
        STORE ch          /* Store the character */
        OUTCHAR ch        /* Write it out */
        LOAD addr         /* Get the current address */
        ADD #1            /* Move to the next one */
        JUMP strout       /* And go back for more */
ret:    RETURN            /* Return from the strout function */

msg:    "hello, world"
ldins:  LOAD 0
addr:   0
ch:     0
```

The Ugliest Program I Ever Wrote

Guy Steele

The Internet Worm



"All the News That's Fit to Print"

The New York Times

Late Edition

New York: Today, partly sunny, milder. High 59-64. Tonight, mostly cloudy. Low 48-54. Tomorrow, cloudy, windy, rain developing. High 57-62. Yesterday: High 58, low 41. Details, page D16.

VOL.CXXXVIII . . . No. 47,679 — Copyright © 1988 The New York Times — NEW YORK, FRIDAY, NOVEMBER 4, 1988 — 50 cents beyond 75 miles from New York City, except on Long Island. — 35 CENTS

The New York Times/[illegible]

Gov. Michael S. Dukakis having his picture taken by a 10-year-old fan at a town meeting in Fairless Hills, Pa., during a tour of the Northeast in which he emphasized the drug problem. Page A10. Vice President Bush addressed supporters a rally in Columbus, Ohio. Less than a week after Mr. Dukakis acknowledged being a liberal, Mr. Bush said yesterday that "this election is not about labels." Page A18.

'Virus' in Military Computers Disrupts Systems Nationwide

By JOHN MARKOFF

In an intrusion that raises questions about the vulnerability of the nation's computers, a Department of Defense network has been disrupted since Wednesday by a rapidly spreading "virus" program apparently introduced by a computer science student.

The program reproduced itself through the computer network, making hundreds of copies in each machine it reached, effectively clogging systems linking thousands of military, corporate and university computers around the nation and preventing them from doing additional work. The virus is thought not to have destroyed any files.

By late yesterday afternoon computer experts were calling the virus the largest assault ever on the nation's computers.

'The Big Issue'

"The big issue is that a relatively benign software program can virtually bring our computing community to its knees and keep it there for some time," said Chuck Cole, deputy computer security manager at Lawrence Livermore Laboratory in Livermore, Calif., one of the sites affected by the intrusion. "The cost is going to be staggering."

Clifford Stoll, a computer security expert at Harvard University, added: "There is not one system manager who is not tearing his hair out. It's causing enormous headaches."

The affected computers carry a tremendous variety of business and research information among military officials, researchers and corporations.

While some sensitive military data are involved, the computers handling the nation's most sensitive secret information, like that on the control of nuclear weapons, are thought not to have been touched by the virus.

Parallel to Biological Virus

Computer viruses are so named because they parallel in the computer world the behavior of biological viruses. A virus is a program, or a set of instructions to a computer, that is either planted on a floppy disk meant to be used with the computer or introduced when the computer is communicating over telephone lines or data networks with other computers.

The programs can copy themselves into the computer's master software, or operating system, usually without calling any attention to themselves. From there, the program can be passed to additional computers.

Depending upon the intent of the software's creator, the program might cause a provocative but otherwise harmless message to appear on the computer's screen. Or it could systematically destroy data in the computer's memory. In this case, the virus program did nothing more than reproduce itself rapidly.

The program was apparently a result of an experiment, which

Continued on Page A21, Column 2

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Inquiry Shows Routine Billing of Government by Industry on Fees, Some Dubious

By JOHN H. CUSHMAN Jr.

Special to The New York Times

WASHINGTON, Nov. 3 — A Pentagon investigation has found that the nation's largest military contractors routinely charge the Defense Department for hundreds of millions of dollars paid to consultants, often without justification.

The report of the investigation said that neither the military's current rules nor the contractors' own policies are adequate to assure that the Government does not improperly pay for privately arranged consulting work. Senior Defense Department officials said the Pentagon was proposing changes to correct the flaws.

While it is not improper for military contractors to use consultants in performing work for the Pentagon, the work must directly benefit the military if it is to be paid for by the Defense Department. Often, Pentagon investigators discovered, this test is not met.

Broader Look at Consultants

The Justice Department's continuing criminal investigation has focused attention on consultants and their role in the designing and selling of weapons, and the Defense Department has been criticized for using consultants too freely. Now the Pentagon's own inves-

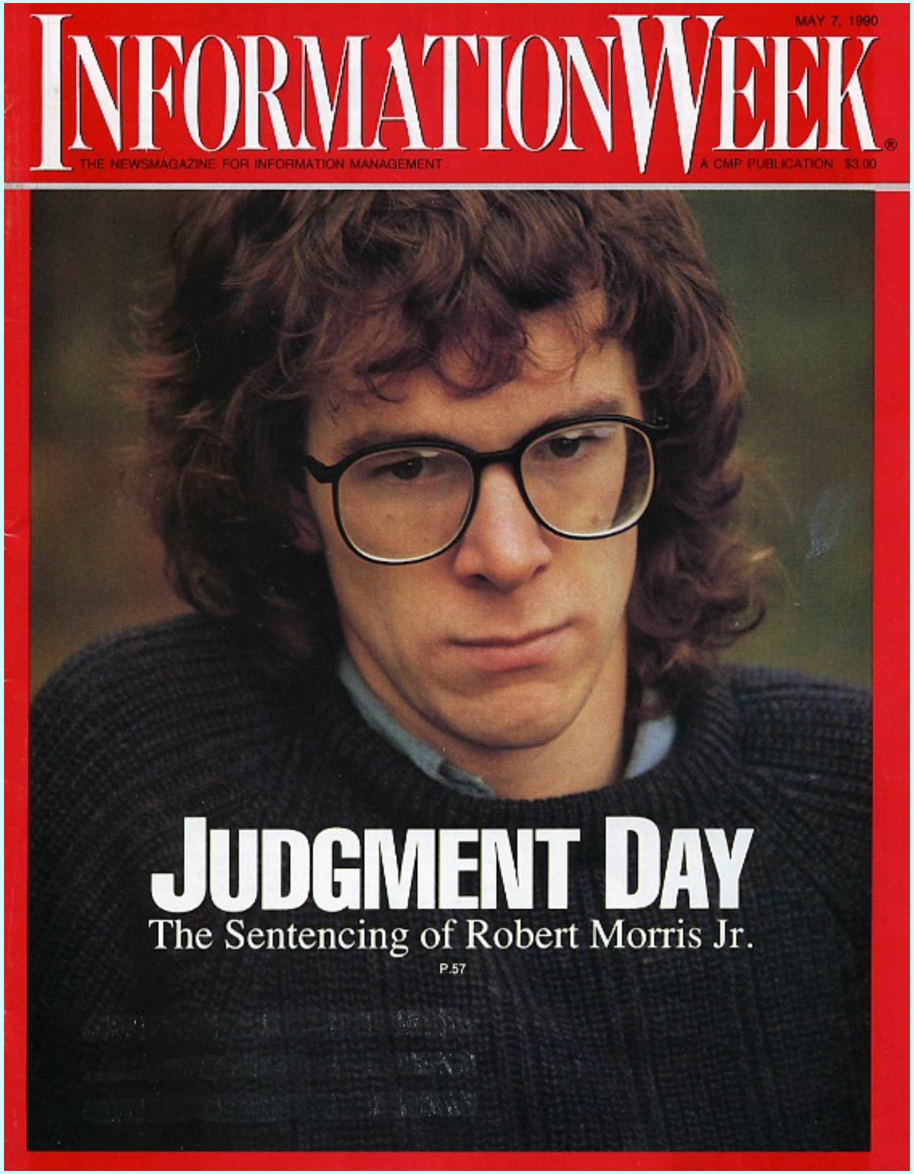
Registration Off Since 1984 Vote

There has been a pronounced decline in the percentage of eligible Americans who are registered to vote, a research group reports.

Nationally, the percentage of eligible Americans who are registered is estimated to be 70.2-percent, down 2.2 points from the 1984 level.

The group's study concluded that in many of the 30 states where final figures are available the decline was among

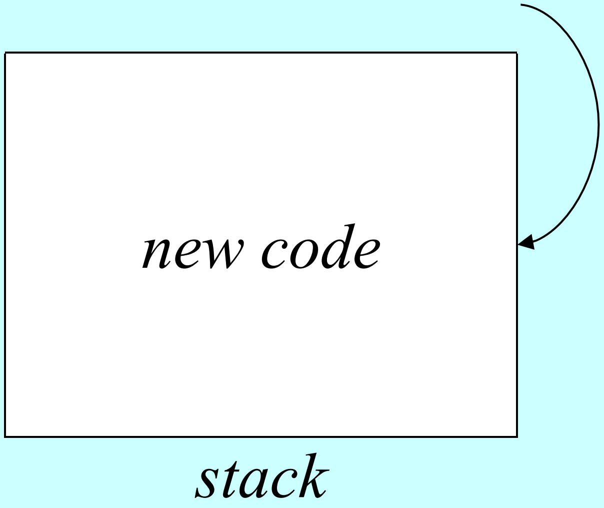
Robert Morris Jr.


MAY 7, 1990
INFORMATIONWEEK®
THE NEWSMAGAZINE FOR INFORMATION MANAGEMENT
A CMP PUBLICATION $3.00
JUDGMENT DAY
The Sentencing of Robert Morris Jr.
P.57

How the Morris Worm Worked

If the user, however, enters a name string that overflows the buffer, the bytes in that name will overwrite the data on the stack.

Now when the function returns, it will jump into the code written as part of the name, thereby executing the worm’s instructions.



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*

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. Supplying the (XR) suffix on a **`LOAD`** or **`STORE`** instruction changes what memory address is referenced.

Hello World: Using the Index Register

```
/*
 * File: HelloWorld3.td
 * --------------------
 * Writes out "hello, world" using the index register.
 */

start:  LOAD #msg         /* Load the address of the string */
        CALL strout       /* Call the function to output a string */
        HALT              /* And halt */

strout: STORE addr        /* Store current address */
        LOADX addr        /* Load that address into the XR */
        LOADX 0(XR)       /* Load the value at that address */
        JUMPZ ret         /* A zero character marks the end */
        STORE ch          /* Store the character */
        OUTCHAR ch        /* Write it out */
        LOAD addr         /* Get the current address */
        ADD #1            /* Move to the next one */
        JUMP strout       /* And go back for more */
ret:    RETURN            /* Return from the strout function */

msg:    "hello, world"
addr:   0
ch:     0
```

The End