University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Typing in a Number

Let's Solve a Problem

Let's see how we can translate
- a task in human terms
- into a program using **LC-3** instructions.

Starting with the task, we shall
- **identify information** that we need to store,
- **assign registers** for stored values
- **draw a flow chart** for the code (roughly at the level of instructions), and
- **write instructions** in human-readable form.

(The actual program is also available to you.)

Overview of Our Task

Let the user type a number
- from 0 to 32767
- using the keyboard, and
- pressing **<Enter>** when done.

Read in the number, convert it to **2's complement**, and store it in memory.

Give errors if
1. a non-digit is pressed, or
2. overflow occurs (> 32767).

Programs are Finite State Machines

Our first question:

What do we need to store?

In other words, what information do we need to have handy in order to solve the problem?

Does this question remind you of FSMs?
- Keys are inputs (including **<Enter>**),
- error messages are outputs, and
- number typed is eventually read out.
- Our program is like an FSM!

What Information Do We Need to Store?

But back to the question:

What do we need to store?

In other words, what information do we need to have handy in order to solve the problem?

1. **the key pressed (one at a time)**
2. **the current value of the user's number (from previous keystrokes)**

Let's Also Store the Value xFFD0

It's also convenient to store xFFD0.

Any idea why?

xFFD0 = –x0030 = '0' (the 0 digit in **ASCII**)

So?

When we want **to convert a digit** typed in **ASCII** into a **2's complement** value,

- **we need to subtract x0030**,
- but we can only subtract x0010 with a single **ADD** instruction.

Dec	Hx	Oct	Html	Chr
32	20	040		Space
33	21	041	!	!
34	22	042	"	"
35	23	043	#	#
36	24	044	$	$
37	25	045	%	%
38	26	046	&	&
39	27	047	'	'
40	28	050	(	(
41	29	051	)	)
42	2A	052	*	*
43	2B	053	+	+
44	2C	054	,	,
45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9

Assign a Register for Each Value that We Store

Finally, we **need a temporary** value for computations.

Let's assign registers.

When we use **GETC** (**TRAP x20**)

- to read a character,
- the keystroke comes back in **R0**, so
- use **R0** for the key pressed.

R1 can be the current value of the number.

R2 can hold xFFD0.

And **R3** can be our temporary.

How Do We Update the Current Value?

When the user presses a key,

how do we update the "current value?"

For example, suppose that

- the user has typed 3, 2, 7, and 6, (in that order),
- so the "current value" is 3276.

If the user presses '7,' we should

- use 3276 and 7
- to calculate 32767.

How?

A Formula for Updating the Current Value

new value = 10 × current value + new digit

Like that?

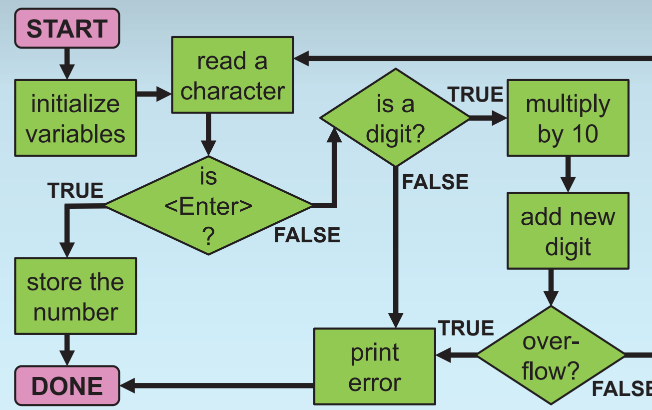
Good, we can use the LC-3 **MULTIPLY** instruction.

Oh.

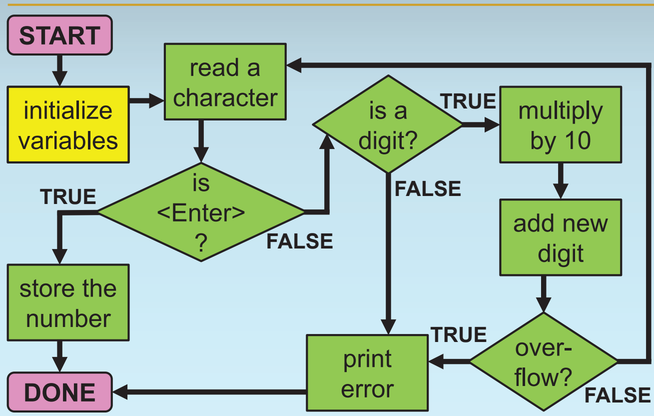
Well, we'll figure it out.

Let's draw a flow chart.

A Flow Chart for Typing in a Number



We're Ready to Write Instructions!



Which Registers Need to Be Initialized?

Here's our table of registers.

R0	key pressed
R1	current value
R2	xFFD0
R3	temporary

R0 will be filled in by **GETC**.

R3 is just a temporary.

We need to set **R2** to **xFFD0**.

What about R1?

(What value should it have before the user presses a key?)

Derive the Initial Value from the Update Formula

Here is our formula for updating:

new value = 10 × current value + new digit

If the user
- first presses 5,
- we want new value to be 5, so

5 = 10 × current value + 5

What should "current value" be? 0

Ok, so we have to initialize **R1** to 0.

Code to Initialize Variables

x3000 LD R2, ______

For R2, we can load the desired value from memory.

Where in memory? After the program. Where's that? We don't know yet.

We need to initialize R2 to xFFD0 and R1 to 0.

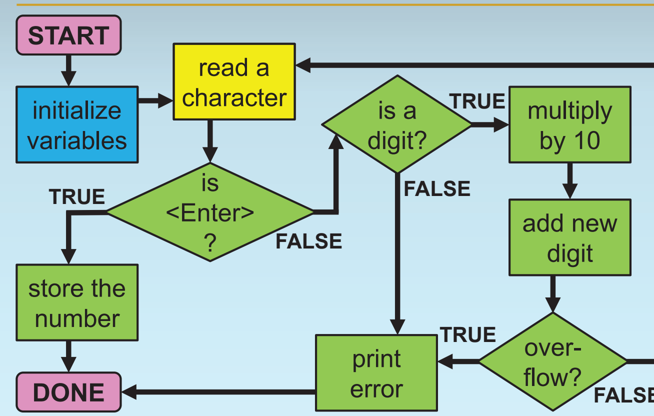
Code to Initialize Variables

x3000 LD R2, ______
x3001 AND R1,R1,#0

For R1, we can use an AND instruction to set it to 0.

We need to initialize R2 to xFFD0 and R1 to 0.

What's Next? Reading a Character from the Keyboard



Code to Read a Character from the Keyboard

```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
```

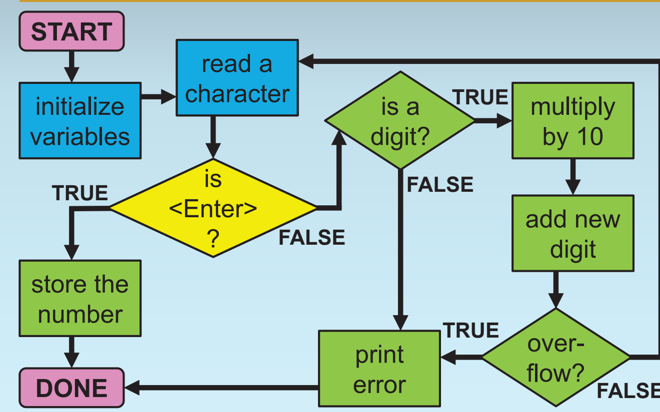
We need to use a TRAP to read a character from the keyboard.

TRAP x20 (GETC) reads a character into R0.

We want the character typed to be echoed to the display (with an OUT trap)!

Why TRAP x21, too?

What's Next? Checking for the <Enter> Key



Code to Check for the <Enter> Key

```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
```

The <Enter> key produces ASCII character #10.

Dec	Hx	Oct	Char	
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)
14	E	016	SO	(shift out)
15	F	017	SI	(shift in)

Subtract #10 to make a comparison.

R0 is the key pressed.

Where does the result go? Discard it (into R3).

Code to Check for the <Enter> Key

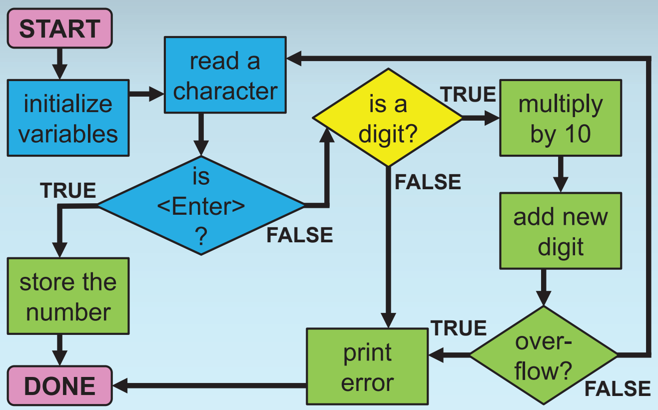
```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
```

The <Enter> key produces ASCII character #10.

Let's branch on zero to the code that stores the number, which is … somewhere. We'll leave the offset blank.

If the result is 0, the key pressed (R0) was <Enter>.

What's Next? Checking for a Digit



Code to Check for a Digit

```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
```

Remember that R2 has negative ASCII digit 0 (xFFD0).

Let's first convert our key to binary, assuming it's a digit.

Code to Check for a Digit

```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
x3007 BRn ______
```

Where to go? Later.

Remember that R2 has negative ASCII digit 0 (xFFD0).

If the result is below 0 (negative), the original character was less than x30, and thus not a digit.

Code to Check for a Digit

```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
x3007 BRn ______
x3008 ADD R3,R0,#-10
```

Now R0 holds the key minus x30. If a digit, 0 to 9.

What about keys greater than '9'?

Let's subtract #10 (and discard the result).

Code to Check for a Digit

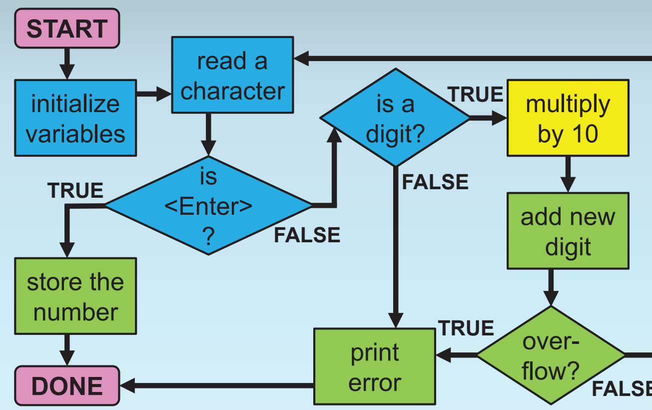
```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
x3007 BRn ______
x3008 ADD R3,R0,#-10
x3009 BRzp ______
```

Now R0 holds the key minus x30. If a digit, 0 to 9.

If the result was equal or greater to 0, the key was greater than '9.'

Where to go? Later.

What's Next? Multiplying by 10



Code to Multiply R1 by 10

```
x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
x3007 BRn ______
x3008 ADD R3,R0,#-10
x3009 BRzp ______
```

Multiply the current value (R1) by 10.

Anyone want to learn ARM now? Or maybe x86? It has "MULT!"

Let me just write some code…

Code to Multiply R1 by 10

```
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
```

Multiply the current value (R1) by 10.

Look good?

Great.

Let's move on?

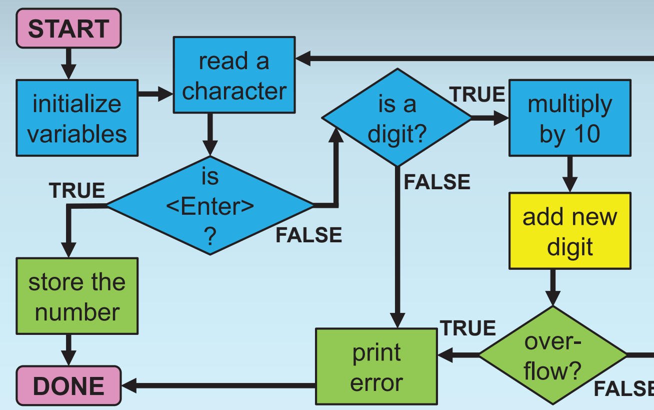
Code to Multiply R1 by 10

```
; This is a comment.
x300A ADD R3,R1,R1
; Now R3 has 2V.
x300B ADD R3,R3,R3
; Now R3 has 4V.
x300C ADD R1,R1,R3
; Now R1 has 5V.
x300D ADD R1,R1,R1
; Now R1 has 10V.
```

Multiply the current value (R1) by 10.

Let's use V to denote the original value of R1.

What's Next? Adding the New Digit



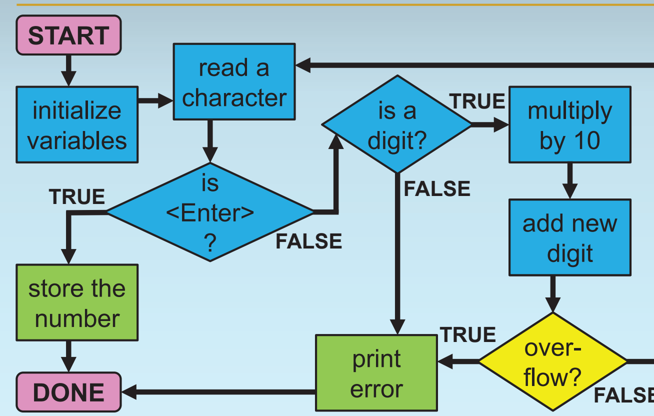
Code to Add the New Digit

```
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
```

Add the new digit into R1.

Where is the new digit? We already computed it and stored it in R0.

What's Next? Checking for Overflow



Download the Code or Printout to See the Overflow Checks

How do we check for overflow?

Unfortunately, it's not so easy.

Checking for overflow requires checking all of the **ADD** instructions.

We won't do that here.

To see the overflow checks, look at the full version provided to you.

Code to Add the New Digit

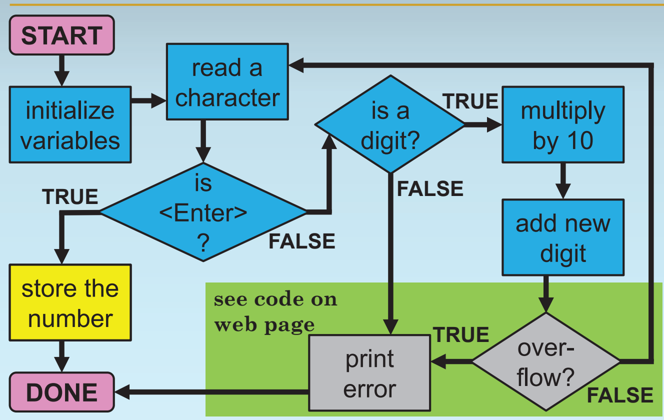
```
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
```

Go get another digit!

How? Use an unconditional branch.

Let's figure out the offset later.

What's Next? Storing the Number



Code to Store the Number

```
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
```

Store the number to memory.

Let's use an ST to store it nearby.

The number is in R1.

Where? Well… later.

Code to End the Program

x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25

And then we're done!

Use a HALT trap (number x25).

Code for Data

x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25
x3012 xFFD0
x3013 place for number

But we still need a couple more things.

First, we need the value xFFD0.

Second, we need a place to store the number.

Here's the Whole Program

x3000 LD R2, ______
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
x3007 BRn ______
x3008 ADD R3,R0,#-10
x3009 BRzp ______
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25
x3012 xFFD0
x3013 place for number

Now for Some Real Fun!

It's time for…

Well, yes, we'll turn them into bits.

But I meant counting!

Almost as exciting as bits.

Help Me Count (Please!)

```
x3000 LD R2, x11
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz ______
x3006 ADD R0,R0,R2
x3007 BRn ______
x3008 ADD R3,R0,#-10
x3009 BRzp ______
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25
x3012 xFFD0
x3013 place for number
```

PC will point here

Help Me Count (Please!)

```
x3000 LD R2, x11
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz xA
x3006 ADD R0,R0,R2
x3007 BRn ______
x3008 ADD R3,R0,#-10
x3009 BRzp ______
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25
x3012 xFFD0
x3013 place for number
```

PC will point here

Help Me Count (Please!)

```
x3000 LD R2, x11
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz xA
x3006 ADD R0,R0,R2
x3007 BRn xC
x3008 ADD R3,R0,#-10
x3009 BRzp ______
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25
x3012 xFFD0
x3013 place for number
```

PC will point here

Help Me Count (Please!)

```
x3000 LD R2, x11
x3001 AND R1,R1,#0
x3002 TRAP x20
x3003 TRAP x21
x3004 ADD R3,R0,#-10
x3005 BRz xA
x3006 ADD R0,R0,R2
x3007 BRn xC
x3008 ADD R3,R0,#-10
x3009 BRzp xA
x300A ADD R3,R1,R1
x300B ADD R3,R3,R3
x300C ADD R1,R1,R3
x300D ADD R1,R1,R1
x300E ADD R1,R1,R0
x300F BRnzp ______
x3010 ST R1, ______
x3011 TRAP x25
x3012 xFFD0
x3013 place for number
```

PC will point here

Help Me Count (Please!)

```
x3000 LD R2, x11          x300A ADD R3,R1,R1
x3001 AND R1,R1,#0        x300B ADD R3,R3,R3
x3002 TRAP x20            x300C ADD R1,R1,R3
x3003 TRAP x21            x300D ADD R1,R1,R1
x3004 ADD R3,R0,#-10      x300E ADD R1,R1,R0
x3005 BRz xA              x300F BRnzp x-E
x3006 ADD R0,R0,R2        x3010 ST R1, ______
x3007 BRn xC              x3011 TRAP x25
x3008 ADD R3,R0,#-10      x3012 xFFD0
x3009 BRzp xA             x3013 place for number
```

PC will point here

Help Me Count (Please!)

```
x3000 LD R2, x11          x300A ADD R3,R1,R1
x3001 AND R1,R1,#0        x300B ADD R3,R3,R3
x3002 TRAP x20            x300C ADD R1,R1,R3
x3003 TRAP x21            x300D ADD R1,R1,R1
x3004 ADD R3,R0,#-10      x300E ADD R1,R1,R0
x3005 BRz xA              x300F BRnzp x-E
x3006 ADD R0,R0,R2        x3010 ST R1, x2
x3007 BRn xC              x3011 TRAP x25
x3008 ADD R3,R0,#-10      x3012 xFFD0
x3009 BRzp xA             x3013 place for number
```

PC will point here

Now We Can Write Bits!

```
x3000 LD R2, x11          x300A ADD R3,R1,R1
x3001 AND R1,R1,#0        x300B ADD R3,R3,R3
x3002 TRAP x20            x300C ADD R1,R1,R3
x3003 TRAP x21            x300D ADD R1,R1,R1
x3004 ADD R3,R0,#-10      x300E ADD R1,R1,R0
x3005 BRz xA              x300F BRnzp x-E
x3006 ADD R0,R0,R2        x3010 ST R1, x2
x3007 BRn xC              x3011 TRAP x25
x3008 ADD R3,R0,#-10      x3012 xFFD0
x3009 BRzp xA             x3013 place for number
```


Encode the Instruction at x3000 into Bits

x3000 LD R2, x11

opcode	destination register	9-bit 2's complement offset
0 0 1 0	0 1 0	0 0 0 0 1 0 0 0 1
LD	R2,	x11

In a program, we **include the spaces between bits** (more readable for humans) and **add a comment**, either "**LD R2,x11**" or "**R2 ← M[PC + x0011]**."

A Binary File Starts with the Starting Address

Also, the starting address, **x3000**, goes first.

For example…

```
0011 0000 0000 0000 ; start at x3000
0010 010 000010001  ; LD R2,x11
; and so forth...
```

opcode	destination register	9-bit 2's complement offset
0 0 1 0	0 1 0	0 0 0 0 1 0 0 0 1
LD	R2,	x11

Encode the Instruction at x3001 into Bits

x3001 AND R1,R1,#0

opcode	destination register	source register 1	mode	5-bit 2's complement
0 1 0 1	0 0 1	0 0 1	1	0 0 0 0 0
AND	R1,	R1,		#0

The Rest is Left to You

I'll leave the rest for you.

I think you can manage it.

Look at the LC-3 encoding table, and write the bits.

Compare your answers with the code provided on the web page.