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

ECE 120: Introduction to Computing

Counting to Ten
(LC-3 Style!)

What is an Instruction Set Architecture (ISA)?

An ISA answers three questions:

1. What operations are possible with instructions?
2. On what operands can each operation be performed?
3. What bits/representation is used to encode instructions?

The answers to these questions define the ISA.

We Discussed Five Aspects of the LC-3 ISA

1. opcodes: **AND**, **AND**, **NOT**
   LD/ST, **LDI/STI**, **LDR/STR**, **LEA**
   BR, **JMP**, **TRAP**
2. data types: **2's complement** (only)
3. addressing modes: **register**, **immediate**, **PC-relative**, **indirect**, **base+offset**
4. condition codes: **negative**, **zero**, **positive**
5. encoding: see P&P p. 119, back of P&P, or LC-3 handout

Let's Count to Ten Together (Using LC-3)

Let's do something exciting with **LC-3**.

Let's count to 10!

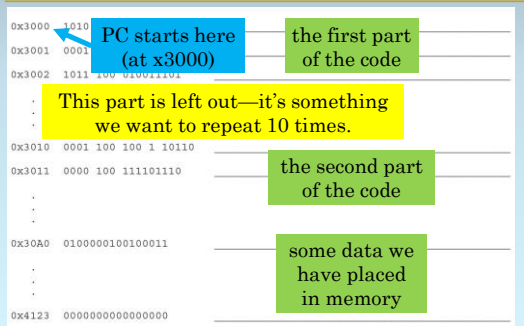
The handout has three versions:

- **PC-relative** addressing (**LD/ST**),
- **indirect** addressing (**LDI/STI**), and
- **base+offset** addressing (**LDR/STR**).

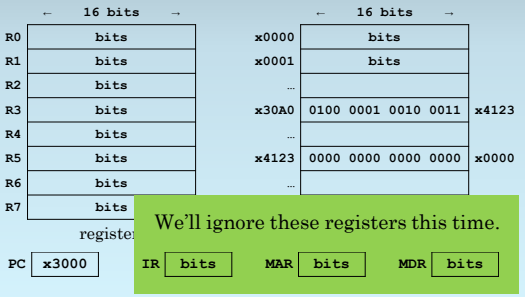
Let's do the **indirect** addressing version together.

Do the others on your own.

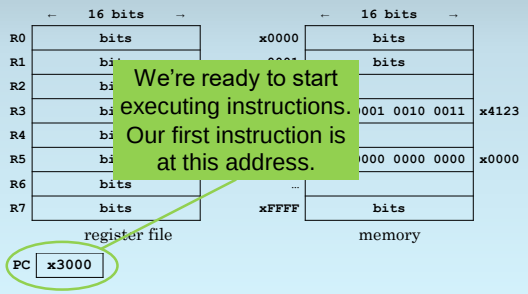
Here's the Code with Indirect Addressing



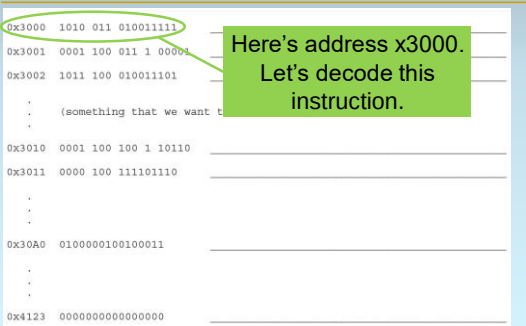
Some Parts of the Datapath for Illustration



Where is the First Instruction?



The First Instruction is at Memory Address x3000



Decode the Instruction at x3000

opcode	destination register	9-bit 2's complement offset
1 0 1 0	0 1 1	0 1 0 0 1 1 1 1 1
LDI	R3,	x09F

R3 ← M[M[PC + x009F]]

Write the Decoded Instruction onto Our Sheet

```
0x3000  1010 011 010011111
0x3001  0001 100 011 1 00001
0x3002  1011 100 010011101
   .
   .    (something that we want to do ten times)
   .
0x3010  0001 100 100 1 10110
0x3011  0000 100 111101110
   .
   .
   .
0x30A0  0100000100100011
   .
   .
   .
0x4123  0000000000000000
```

LDI R3,x09F

Now, let's execute it!

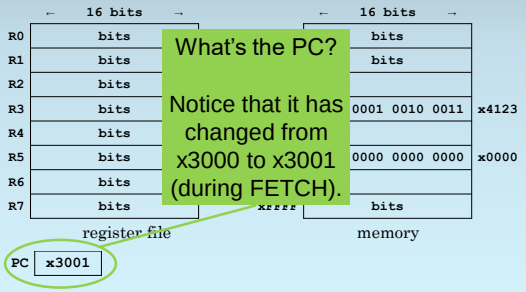
Execute the LDI Instruction

opcode	destination register	9-bit 2's complement offset
1 0 1 0	0 1 1	0 1 0 0 1 1 1 1 1
LDI	R3,	x09F

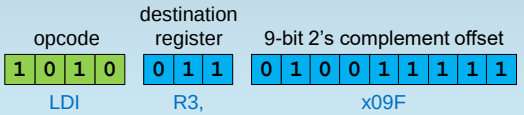
R3 ← M[M[PC + x009F]]

What is the first memory address?

Read the Value of the PC



Execute the LDI Instruction

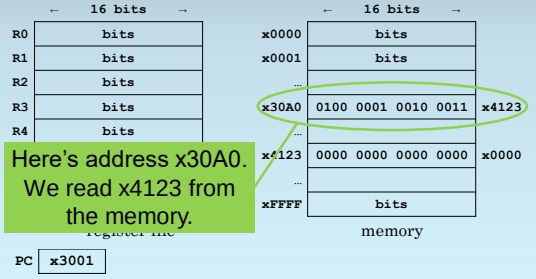


`R3 ← M[M[PC + x009F]]`

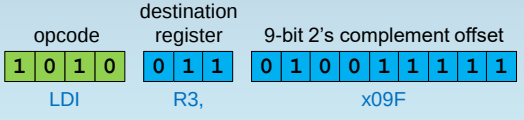
What is the first memory address?

x3001 + x009F gives us x30A0.

Read Memory at x30A0



Execute the LDI Instruction

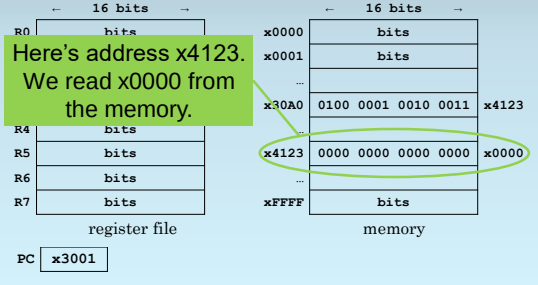


`R3 ← M[M[PC + x009F]]`

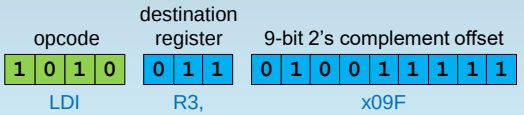
`M[PC + x009F]` returns **x4123**.

Next, read memory address x4123.

Read Memory at x4123



Execute the LDI Instruction

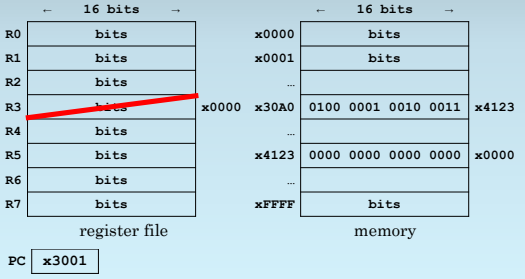


R3 ← M[M[PC + x009F]]

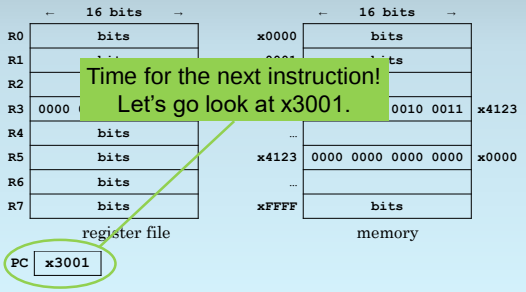
M[M[PC + x009F]] returns **x0000**.

Finally, write x0000 into R3.

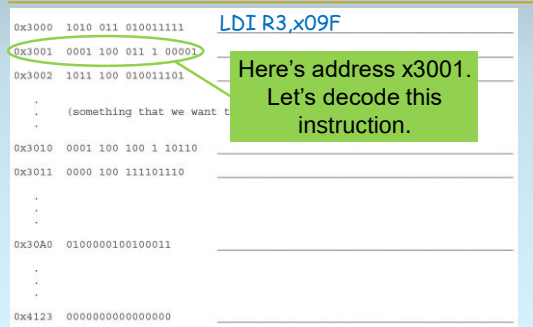
Write x0000 into R3



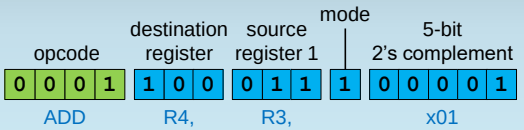
Time for Another Instruction Fetch!



The Second Instruction is at Memory Address x3001



Decode the Instruction at x3001



R4 ← R3 + 1

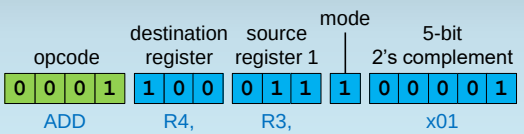
Write the Decoded Instruction onto Our Sheet

```
0x3000  1010 011 010011111
0x3001  0001 100 011 1 00001
0x3002  1011 100 010011101
   .
   .    (something that we want to do ten times)
   .
0x3010  0001 100 100 1 10110
0x3011  0000 100 111101110
   .
   .
   .
0x30A0  0100000100100011
   .
   .
   .
0x4123  0000000000000000
```

LDI R3,x09F
ADD R4,R3,x01

Now, let's execute it!

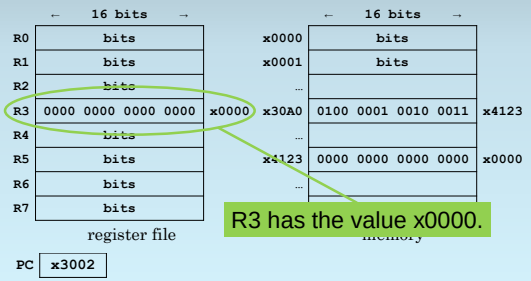
Execute the ADD Instruction



R4 ← R3 + 1

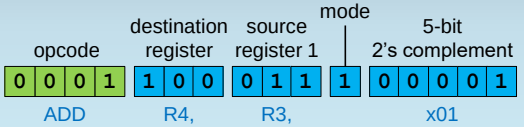
What's in R3?

What's in R3?



R3 has the value x0000.

Execute the ADD Instruction

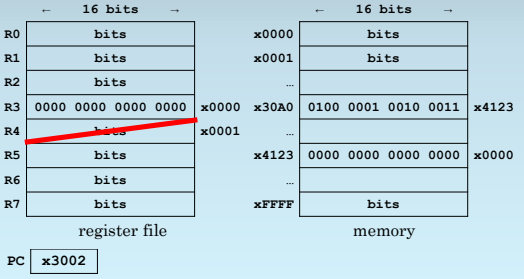


R4 ← R3 + 1

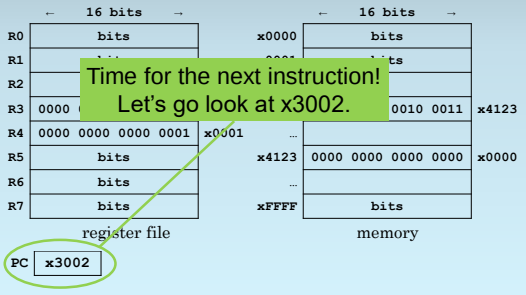
What's in R3? x0000

Add 1 to get x0001, then write it to R4.

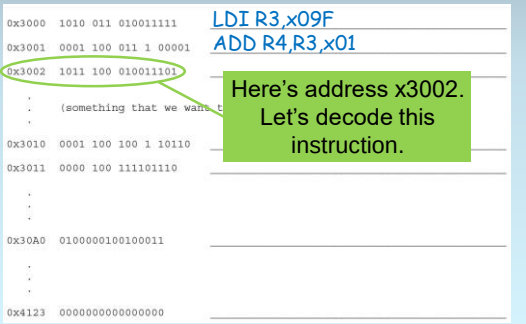
Write x0001 into R4



Time for Another Instruction Fetch!



The Third Instruction is at Memory Address x3002



Decode the Instruction at x3002

opcode	source register	9-bit 2's complement offset
1 0 1 1	1 0 0	0 1 0 0 1 1 1 0 1
STI	R4,	x09D

M[M[PC + x009D]] ← R4

Write the Decoded Instruction onto Our Sheet

```
0x3000  1010 011 010011111
0x3001  0001 100 011 1 00001
0x3002  1011 100 010011101
   .
   .    (something that we want to
   .
0x3010  0001 100 100 1 10110
0x3011  0000 100 111101110
   .
   .
   .
0x30A0  0100000100100011
   .
   .
   .
0x4123  0000000000000000
```

LDI R3,x09F
ADD R4,R3,x01
STI R4,x09D

Now, let's execute it!

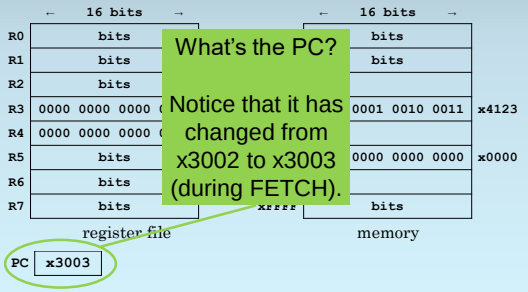
Execute the STI Instruction

opcode	source register	9-bit 2's complement offset
1 0 1 1	1 0 0	0 1 0 0 1 1 1 0 1
STI	R4,	x09D

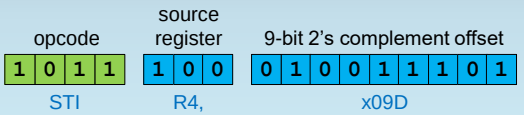
M[M[PC + x009D]] ← R4

What is the first memory address?

Read the Value of the PC



Execute the STI Instruction

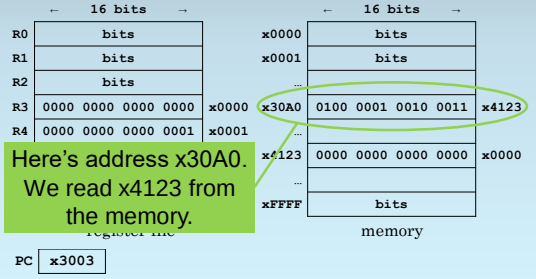


`M[M[PC + x009D]] ← R4`

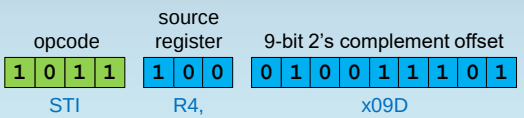
What is the first memory address?

x3003 + x009D again gives us x30A0.

Read Memory at x30A0



Execute the STI Instruction

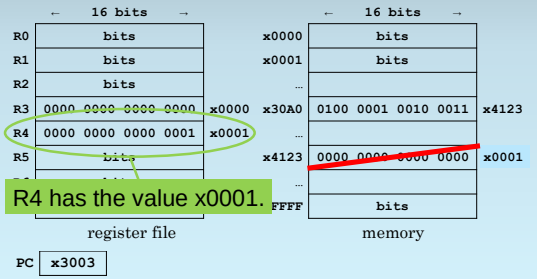


`M[M[PC + x009D]] ← R4`

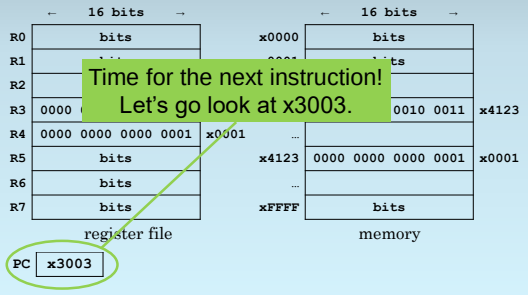
`M[PC + x009D]` returns **x4123**.

Next, write R4 to memory address x4123.

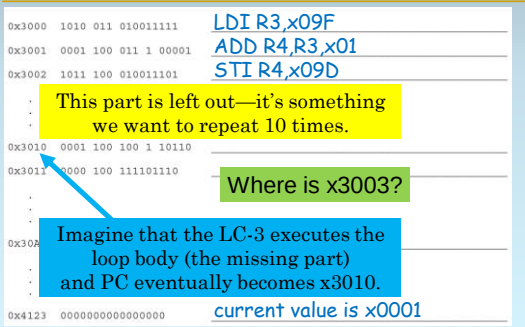
Store the Bits from R4 at Memory Address x4123



Time for Another Instruction Fetch!



The LC-3 Has Reached the Loop Body



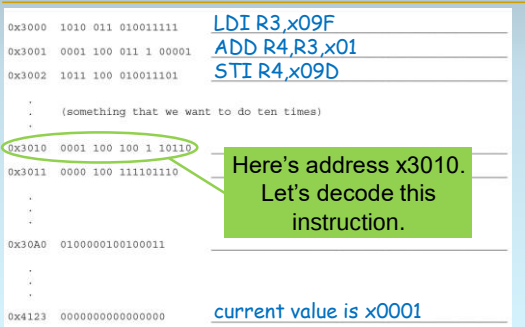
Let's Track Some Values for the Loop Body

Let's make a table of loop body executions.

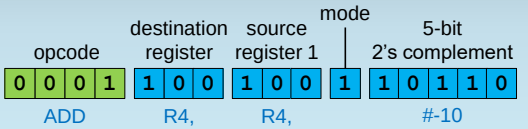
What is R4 during the first execution?

loop body execution #	R4 during loop body	R4 at BRn
1	1	

The Next Instruction is at Memory Address x3010



Decode the Instruction at x3010



R4 ← R4 - #10

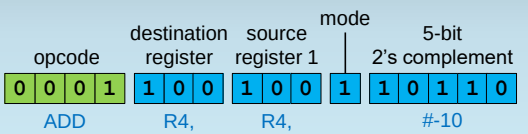
"#" means decimal in LC-3 tool notation

Write the Decoded Instruction onto Our Sheet

```
0x3000  1010 011 010011111      LDI R3,x09F
0x3001  0001 100 011 1 00001    ADD R4,R3,x01
0x3002  1011 100 010011101      STI R4,x09D
   .
   .    (something that we want to do ten times)
   .
0x3010  0001 100 100 1 10110    ADD R4,R4,#-10
0x3011  0000 100 111101110      
   .
   .
   .
0x30A0  0100000100100011        
   .
   .
   .
0x4123  0000000000000000        current value is x0001
```

Now, let's execute it!

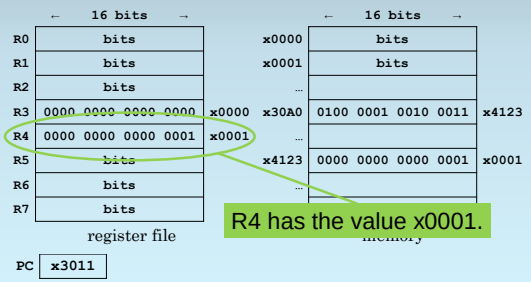
Execute the ADD Instruction



R4 ← R4 - #10

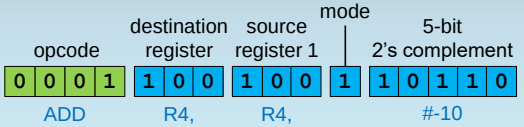
What's in R4?

What's in R4?



R4 has the value x0001.

Execute the ADD Instruction

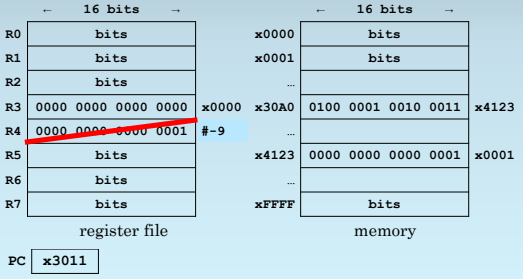


R4 ← R4 + #10

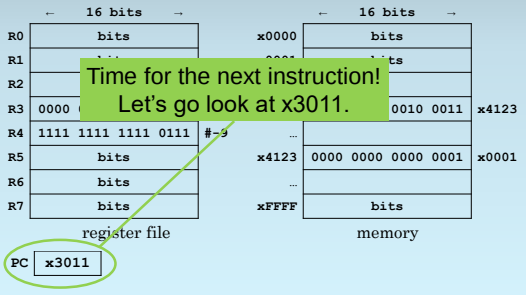
What's in R4? x0001

Subtract 10 to get #-9, then write it back to R4.

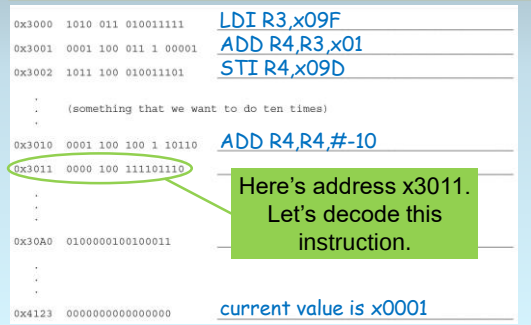
Write #-9 to R4



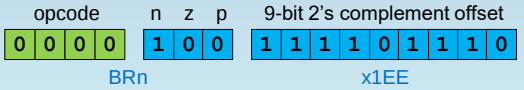
Time for Another Instruction Fetch!



The Next Instruction is at Memory Address x3011



Decode the Instruction at x3011



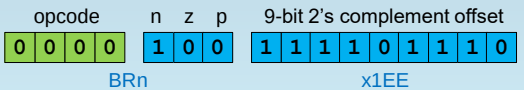
nN: PC ← PC + xFFEE

Write the Decoded Instruction onto Our Sheet

```
0x3000  1010 011 010011111      LDI R3,x09F
0x3001  0001 100 011 1 00001    ADD R4,R3,x01
0x3002  1011 100 010011101      STI R4,x09D
   .
   .    (something that we want to do ten times)
   .
0x3010  0001 100 100 1 10110    ADD R4,R4,#-10
0x3011  0000 100 111101110      BRn x1EE
   .
   .
   .
0x30A0  0100000100100011
   .
   .
   .
0x4123  0000000000000000        current value is x0001
```

Now, let's execute it!

Execute the BR Instruction



nN: PC ← PC + xFFEE

What was last written to the register file?

R4, with value #-9. That's negative.

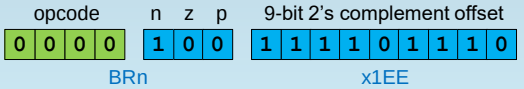
Fill in the Value of R4 at the BRn

R4 has #-9 when we reach BRn.

Let's add it to our table.

loop body execution #	R4 during loop body	R4 at BRn
1	1	-9

Execute the BR Instruction

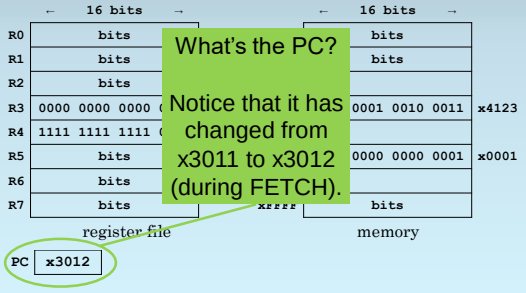


nN: PC ← PC + xFFEE

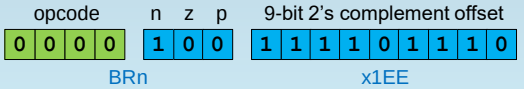
So the branch is taken (the PC changes).

What is the current value of PC?

Read the Value of the PC



Execute the BR Instruction

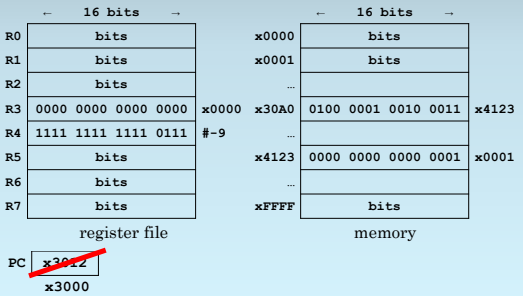


nN: PC ← PC + xFFEE

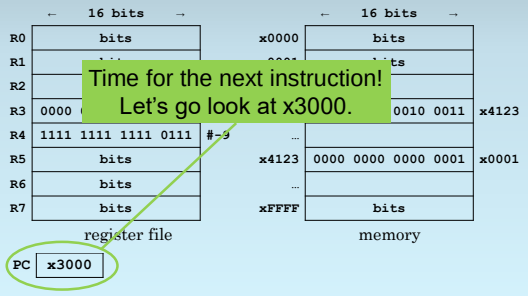
What is the current value of PC? x3012

Adding xFFEE, we get x3000.
So we write x3000 to the PC.

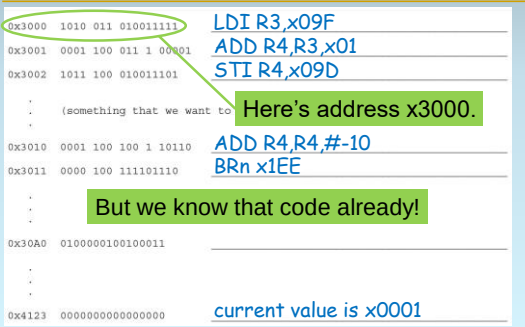
Set the PC to x3000



Time for Another Instruction Fetch!



We Have Returned to the Start of the Loop



The PC-Relative Addresses Do Not Change

Here is the **RTL** for the first three instructions.

```
x3000 R3 ← M[M[PC + x009F]]
x3001 R4 ← R3 + 1
x3002 M[M[PC + x009D]] ← R4
```

Since the **values of PC**
- **depend on the instruction addresses**,
- **the same calculations** are performed
- each time this code executes.

Assume that Nothing Changed Address x30A0

Thus, we can simplify the **RTL** slightly.

```
x3000 R3 ← M[M[x30A0]]
x3001 R4 ← R3 + 1
x3002 M[M[x30A0]] ← R4
```

Let's **assume that** the bits stored at memory **address x30A0** have **not changed**.

So we can also simplify by replacing **M[x30A0]** with **x4123** in both **LDI** and **STI**.

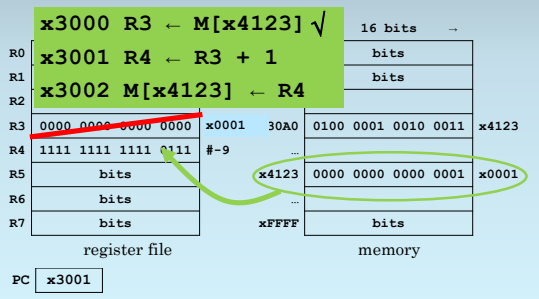
Ready to Execute the RTL on the Datapath

Simplifying the **RTL** again, we obtain:

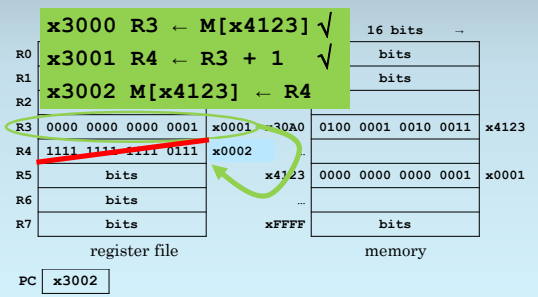
```
x3000 R3 ← M[x4123]
x3001 R4 ← R3 + 1
x3002 M[x4123] ← R4
```

Let's go **execute these three instructions** on the datapath now.

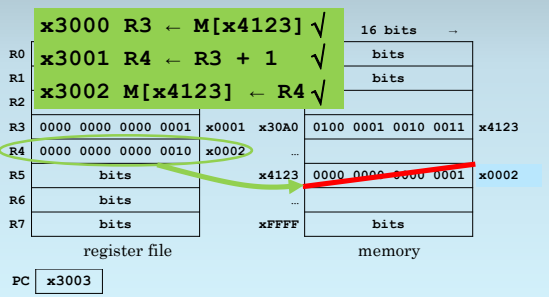
Second Execution of the First Section of Code



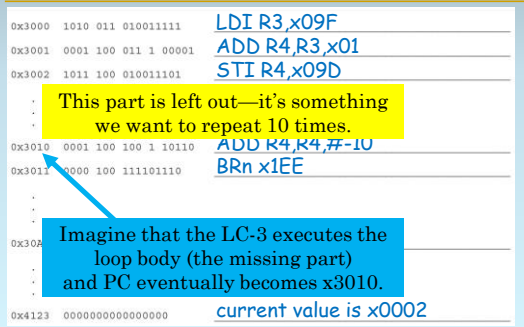
Second Execution of the First Section of Code



Second Execution of the First Section of Code



The LC-3 Has Again Reached the Loop Body



Fill in the Value of R4 for the Second Loop Body Execution

What is R4 during the second loop body execution?

loop body execution #	R4 during loop body	R4 at BRn
1	1	-9
2	2	

The PC-Relative Addresses Do Not Change

Here is the **RTL** for the code after the loop body.

```
x3010 R4 ← R4 - #10
x3011 nN: PC ← PC + xFFEE
```

Again, the **value of PC** (on the right)
- **depends on the instruction address**, so
- **the same calculation** is performed
- each time this code executes.

Ready to Execute the RTL on the Datapath

Thus, we can simplify the **RTL** slightly.

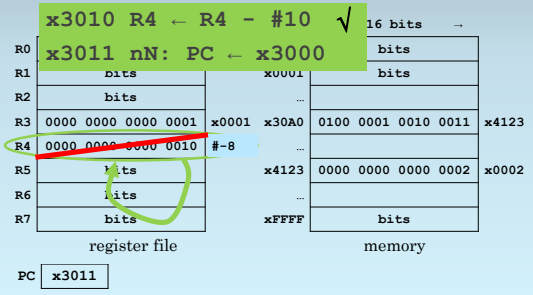
```
x3010 R4 ← R4 - #10
x3011 nN: PC ← x3000
```

Let's go **execute these two instructions** on the datapath now.

Second Execution of the Second Section of Code

x3010 R4 ← R4 + #10 √

x3011 nN: PC ← x3000



Fill in the Value of R4 at the BRn

R4 has #-8 when we reach BRn.

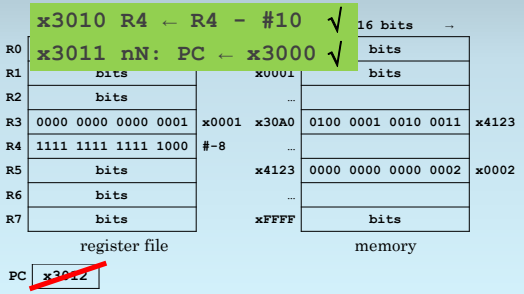
Let's add it to our table.

loop body execution #	R4 during loop body	R4 at BRn
1	1	-9
2	2	-8

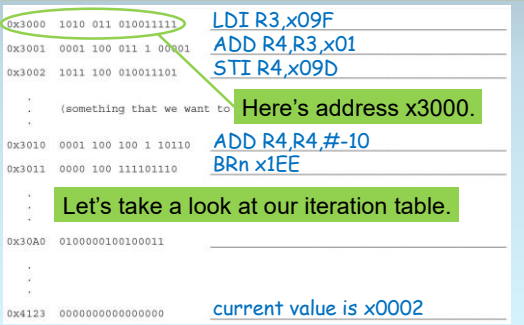
Second Execution of the Second Section of Code

x3010 R4 ← R4 + #10 √

x3011 nN: PC ← x3000 √



We Have Returned to the Start of the Loop (Again!)



Let's Generalize Our Table Values

R4 counts up with the loop body execution #.

When does R4 get to 0 (non-negative)?

loop body execution #	R4 during loop body	R4 at BRn
1	1	-9
2	2	-8
…	…	…
10	10	0

The Loop Ends After Ten Iterations

In other words,

- **after the tenth loop body iteration**
- the **branch is not taken**,
- and the **PC** remains **x3012**.

Guess what the LC-3 does.

Executes another instruction!

But we're going to stop.

Some Questions for You

1. Why is there a 0 stored at x4123?
2. How many times does the loop body execute if we start M[x4123] at 5?
3. How many times does the loop body execute if we start M[x4123] at -5?
4. How many times does the loop body execute if we start M[x4123] at 25?

More Questions for You

5. What if we leave M[x4123] as "bits" (set no value there)?
6. What happens if we change the value at x30A0 to x3141?
7. What happens if the loop body sets R4 to 0?

A Reference Copy of the Code

```
0x3000  1010 011 010011111      LDI R3,x09F
0x3001  0001 100 011 1 00001    ADD R4,R3,x01
0x3002  1011 100 010011101      STI R4,x09D

   .
   .    (something that we want to do ten times)
   .

0x3010  0001 100 100 1 10110    ADD R4,R4,#-10
0x3011  0000 100 111101110      BRn x1EE

   .
   .
   .

0x30A0  0100000100100011        x4123 (data)

   .
   .
   .

0x4123  0000000000000000        x0000 (data)
```