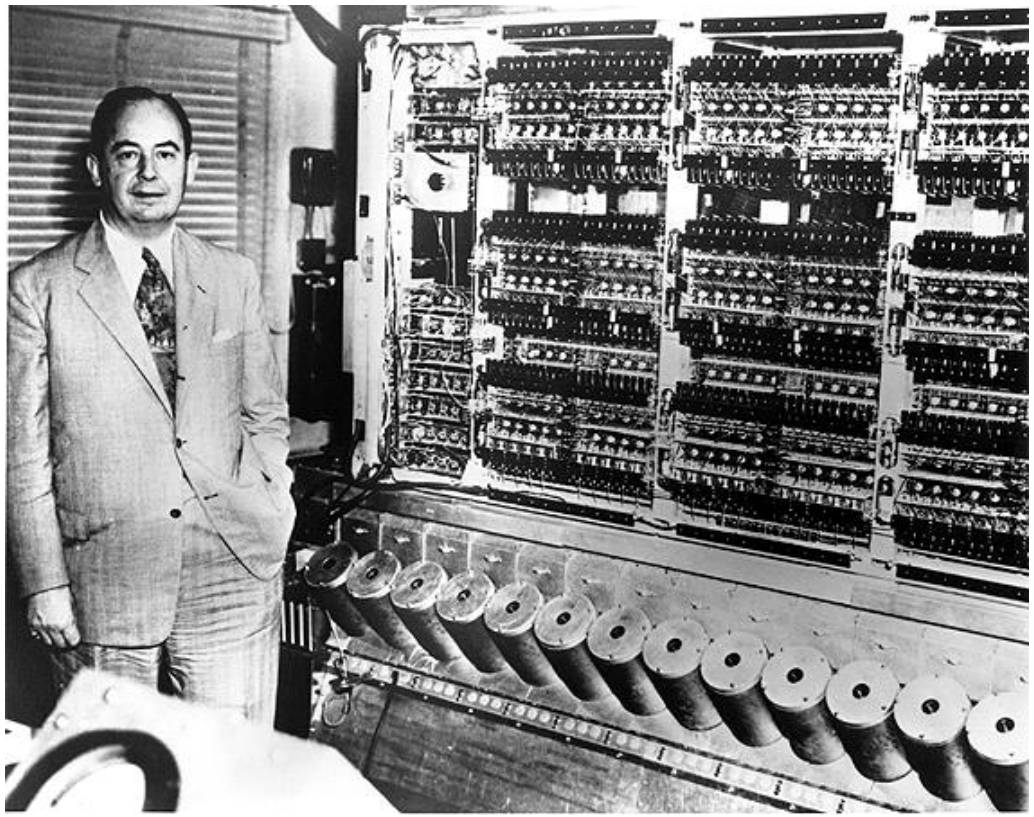
Little Computer 3 (LC3)
A Simple Architecture

Text: Introduction to Computer Systems : Sections 4.1, 4.3, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.2

Computer vs. Adding Machine



What is the difference?

Computers are Programmable

```
sub Calculator()
    sub addition(self, other)
        return self + other
    end

    /** Create a list of 2 numbers. */
    sub makeFraction(numerator, denominator)
        return [numerator, denominator]
    end

    /** Warning: Destroys original fraction! */
    sub multiplyFracs(frac, otherFrac)
        frac[0] *= otherFrac[0]
        frac[1] *= otherFrac[1]
        return frac
    end
end

sub InfinityCalculator()
    inh
    /**
    sub
    end
end
```

Method

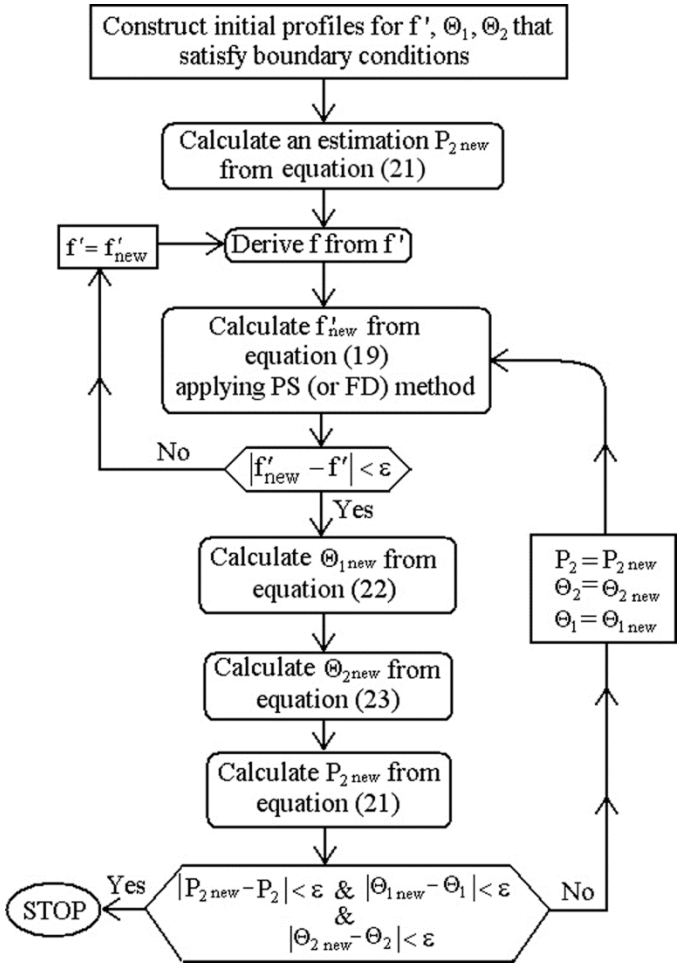
Class

```
function check(n)
  {   // check if the number n is a prime
    var factor; // if the checked number is not a prime, this is its first factor
    var c;
    factor = 0;
    // try to divide the checked number by all numbers till its square root
    for (c=2 ; (c <= Math.sqrt(n)) ; c++)
       {
        if (n%c == 0)    // is n divisible by c ?
           {factor = c;  break}
       }
    return (factor);
  }  // end of check function

function communicate()
  {  // communicate with the user
    var i;       // i is the checked number
    var factor; // if the checked number is not a prime, this is its first factor
    i = document.primetest.number.value;      // get the checked number
    // is it a valid input?
    if ((isNaN(i)) || (i <= 0) || (Math.floor(i) != i))
      {alert ("The checked object should be a whole positive number")} ;
    else
      {
       factor = check (i);
       if (factor == 0)
          {alert (i + " is a prime")} ;
       else
          {alert (i + " is not a prime, " + i + "=" + factor + "X" + i/factor) }
      }
```

```
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32   PROC            ; procedure begins here
        CMP   AX,97     ; compare AX to 97
        JL    DONE      ; if less, jump to DONE
        CMP   AX,122    ; compare AX to 122
        JG    DONE      ; if greater, jump to DONE
        SUB   AX,32     ; subtract 32 from AX
DONE:   RET             ; return to main program
SUB32   ENDP            ; procedure ends here
```



What is a computer program?

Computer Program Definition (Wikipedia)

- A **computer program**, or just a **program**, is a sequence of instructions, written to perform a specified task on a computer.
- A computer requires programs to function, typically executing the program's instructions in a central processor.
- The program has an executable form that the computer can use directly to execute the instructions.
- The same program in its human-readable source code form, from which executable programs are derived (e.g., compiled)…

Executable Instructions

- Like a sentence with a verb and a noun and an object
- What to do
  - add, subtract, move, …
- What to do it on
  - Data in the instruction?
  - Data in memory?
  - Data from a file?
- Where to put the result

```
while (a=getc()) {
    a=a+1;
    sum=sum+a;
}
```

“Instruction Set Architecture” (ISA)

```
SUB32   PROC          ; procedure begins here
        CMP   AX,97   ; compare AX to 97
        JL    DONE    ; if less, jump to DONE
        CMP   AX,122  ; compare AX to 122
        JG    DONE    ; if greater, jump to DONE
        SUB   AX,32   ; subtract 32 from AX
DONE:   RET           ; return to main program
SUB32   ENDP          ; procedure ends here
```

I S A

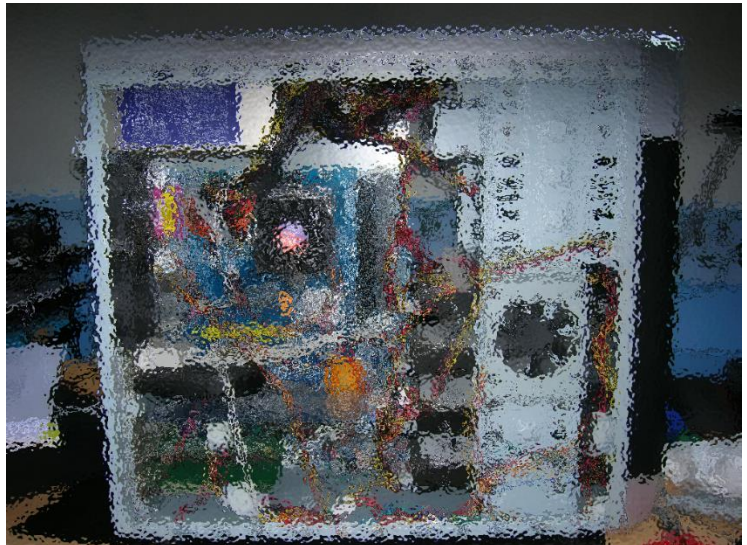
ISA Contents

- The instructions the hardware recognizes
  - add, move, get, …
- The data types the instructions can work on
  - two's complement binary, ascii character, unsigned binary, etc.
- The data the instructions can work on
  - Registers
  - Memory
- The external interfaces supported by the instructions
  - File I/O
  - Exception Handling and Interrupts

Little Computer 3 (LC3)

```
.orig x3000
ADD R1,R1,9          ; R1 <- R1+9
ADD R2,R2,10         ; R2 <- R2+10
ADD R3,R3,11         ; R3 <- R3+11
ADD R1,R2,R3         ; R1 <- R3+R3 = 10 + 11 = 21
ADD R3,R1,R3         ; R3 <- R1+R3 = 21 + 11 = 32
HALT
.end
```

LC3 ISA (Text, Appendix A)



LC3 -> X86 Translator

LC3 Data Types

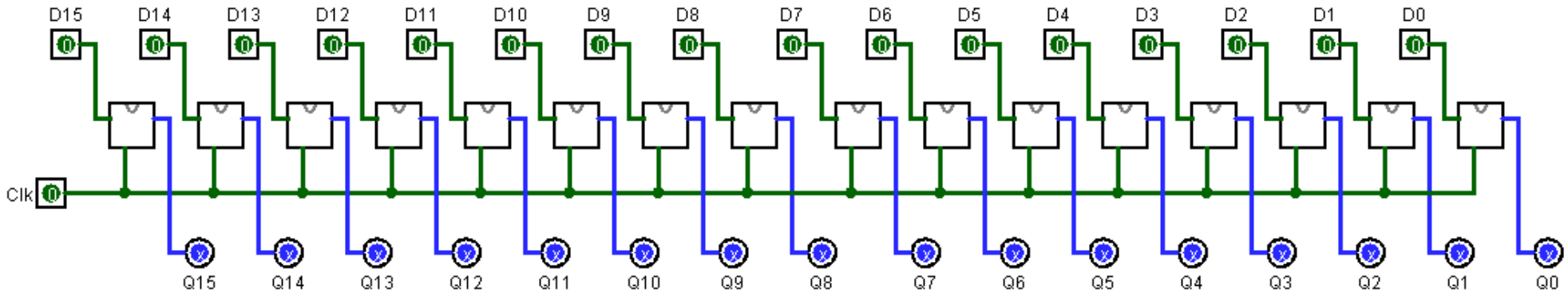
- All data is represented as a 16 bit word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	0	0	0	0	0	1	1	1	0	0

Also represented with 4 hex digits, e.g. 0x281C

- A word can be interpreted as:
  - a 16 bit unsigned binary number: $0 \leftrightarrow (2^{16}-1)=65,535$
  - a 16 bit two's complement binary number: $-32,768 \leftrightarrow 32,767$
  - Two ASCII characters
  - Absolute Address in Memory: 0x0000 ↔ 0xFFFF
  - LC3 Instruction

LC3 General Purpose Registers

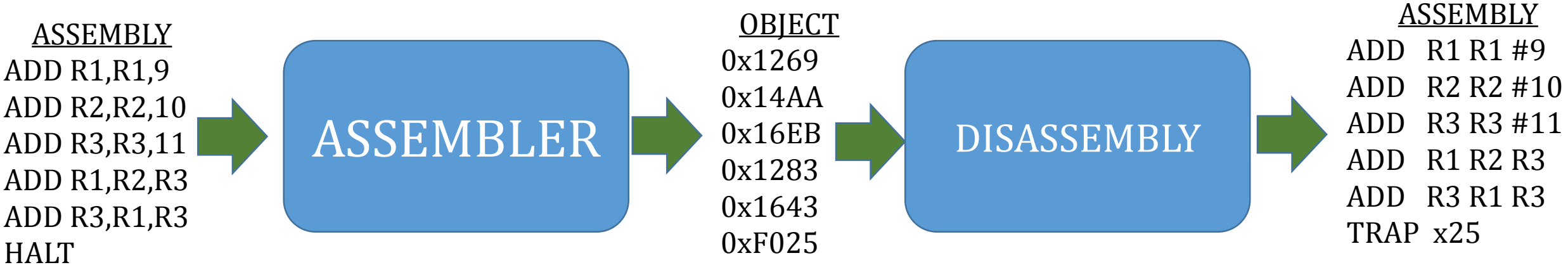


- 8 Registers labeled R0, R1, … R7
- Each register16 bits (1 word, four hex chars) wide
- Fast Read/Write
- All computation from/to registers
- No explicit data type!
- Value undefined (X) until set

R0	0x0000
R1	0x4AC3
R2	0xXXXX
R3	0xFFFF
R4	0x0001
R5	0xXXXX
R6	0xXXXX
R7	0xXXXX

LC3 Instructions

- Smallest (Atomic) directive to LC3 “hardware”
- Two Flavors
  - Man-readable “Assembly”
  - Machine Readable “Object Code” or “Machine Code” or “Binary”
- Translation…



LC3 Instruction

Assembly

- Single Line of Text
- Label – optional, Col. 1 text
- Operation Code (OPCODE)
  - Type of instruction
  - 2/3 letter mnemonic
- Operands depend on opcode
- Comment – optional follows ";"

Object

- 1 word (16 bytes)
- No labels
- Operation Code (OPCODE)
  - Type of instruction
  - 4 bit binary number (0-F)
- Operands depend on opcode
- No comments

LC3 Assembly Operands

- Operands separated by commas ","
- Registers: R0, R1, … , R7 (or r0, r1, …, r7)
- Literal:
  - Decimal number: <decimal digits> e.g. 10, 321, -7, ...
    - Optional "#" prefix, #10, #321, #-7
  - Hexadecimal number: x<hex digits> e.g. x10, xFFFF
  - String: "<characters>" e.g. "Enter a number"
- Label Reference:
  - Any string not a register, literal, or "reserved" word (opcode mnemonic)
  - Must appear as a label in some other instruction

LC3 Operand Conventions

- Destination – where to put the result of this instruction
  - Always (almost) First Operand
- Source(s) – Where to get the data for this instruction
  - Always (almost) Second (& Third Operands)
- e.g. `ADD R1,R2,R3 ; R1 ← R2 + R3`

- Source can match destination
  - e.g. `ADD R1,R1,R3 ; Increment R1 by R3`
- Both sources may be the same
  - e.g. `ADD R2,R1,R1 ; R2 ← 2 x R1`

ADD Instruction (Register Operands)

Assembly

- Mnemonic: ADD
- Destination Register (Rz)
- Source Register 1 (Ra)
- Source Register 2 (Rb)

e.g. `add R1,r2,r3`

Object

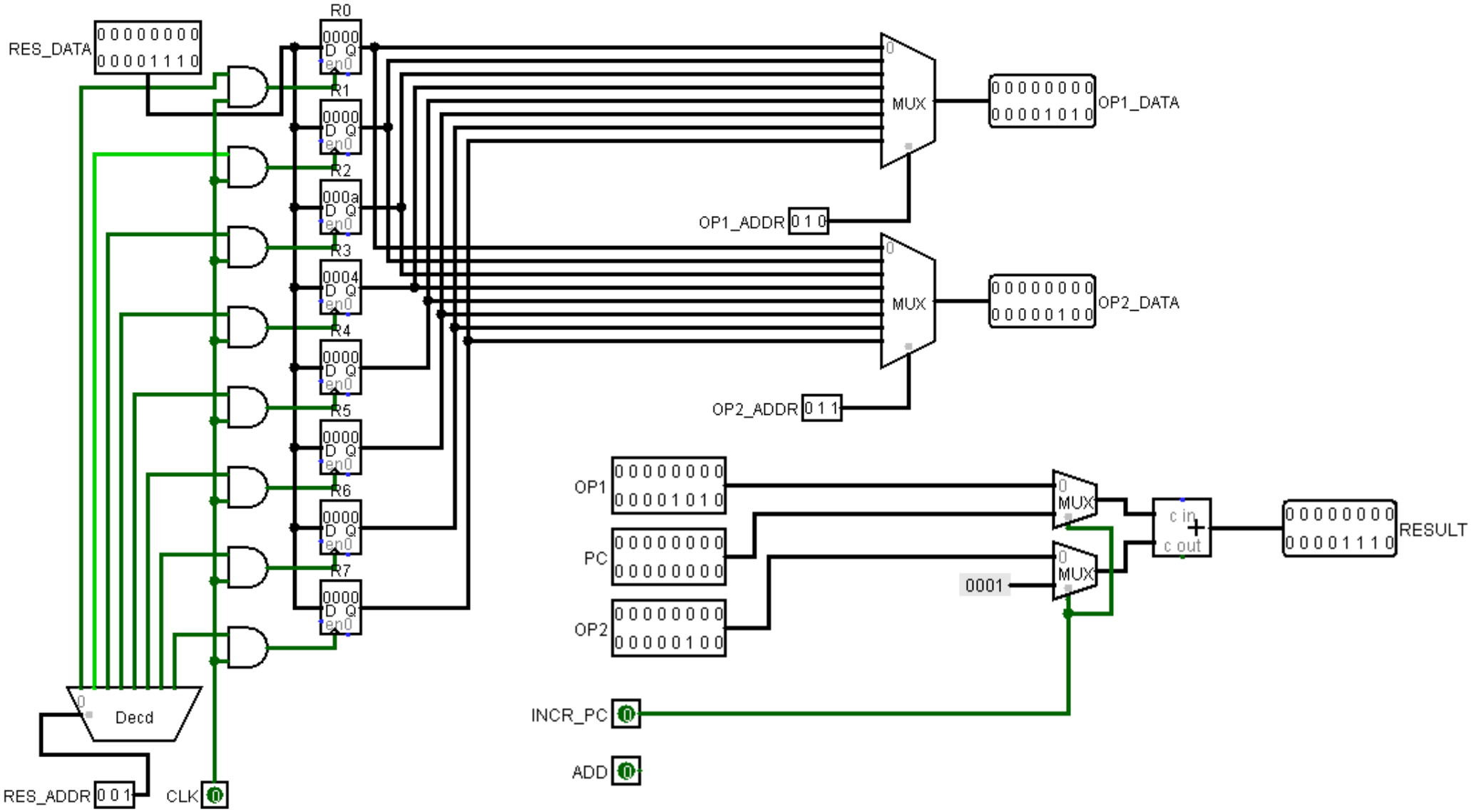
- Opcode: 0001 (0x1)
- DR (3 bit subfield)
- SR1 (3 bit subfield)
- Pad 0b000
- SR2 (3 bit subfield)

e.g. 0b0001 001 010 000 011

=0x1283

Adds value in SR1(Ra) to value in SR2(Rb), stores result in DR (Rz)

Hardware for ADD: `add r1,r2,r3`



ADD (Reg/Immediate Operands)

Assembly

- Mnemonic: ADD
- Destination Register (Rz)
- Source Register 1 (Ra)

- -16 ≤ Literal value ≤ 15

e.g. `add R1,r2,#10`

Object

- Opcode: 0001 (0x1)
- DR (3 bit subfield)
- SR1 (3 bit subfield)
- Literal Flag: 0b1
- IMM5: 5 bit 2's comp binary

e.g. 0b0001 001 010 1 01010 =0x12AA

Adds value in SR1(Ra) to literal value, stores result in DR (Rz)

LC3 AND Instruction (Register Operands)

Assembly

- Mnemonic: AND
- Destination Register (Rz)
- Source Register 1 (Ra)
- Source Register 2 (Rb)

e.g. `and R1,r2,r3`

Object

- Opcode: 0101 (0x5)
- DR (3 bit subfield)
- SR1 (3 bit subfield)
- Pad 0b000
- SR2 (3 bit subfield)

e.g. 0b0101 001 010 000 011

=0x5283

bitwise ands value in SR1(Ra) with value in SR2(Rb), stores result in DR (Rz)

AND (Reg/Immediate Operands)

Assembly

- Mnemonic: AND
- Destination Register (Rz)
- Source Register 1 (Ra)

- -16 ≤ Literal value ≤ 15

e.g. `add R1,r2,#10`

Object

- Opcode: 0101 (0x5)
- DR (3 bit subfield)
- SR1 (3 bit subfield)
- Literal Flag: 0b1
- IMM5: 5 bit 2's comp binary

e.g. 0b0101 001 010 1 01010 =0x52AA

Bitwise ands value in SR1(Ra) with literal value, stores result in DR (Rz)

NOT Instruction

Assembly

- Mnemonic: NOT
- Destination Register (Rz)
- Source Register 1 (Ra)

e.g. `not R1,r2`

Object

- Opcode: 1001 (0x9)
- DR (3 bit subfield)
- SR1 (3 bit subfield)
- Pad 0b111111

e.g. 0b1001 001 010 111111
=0x92BF

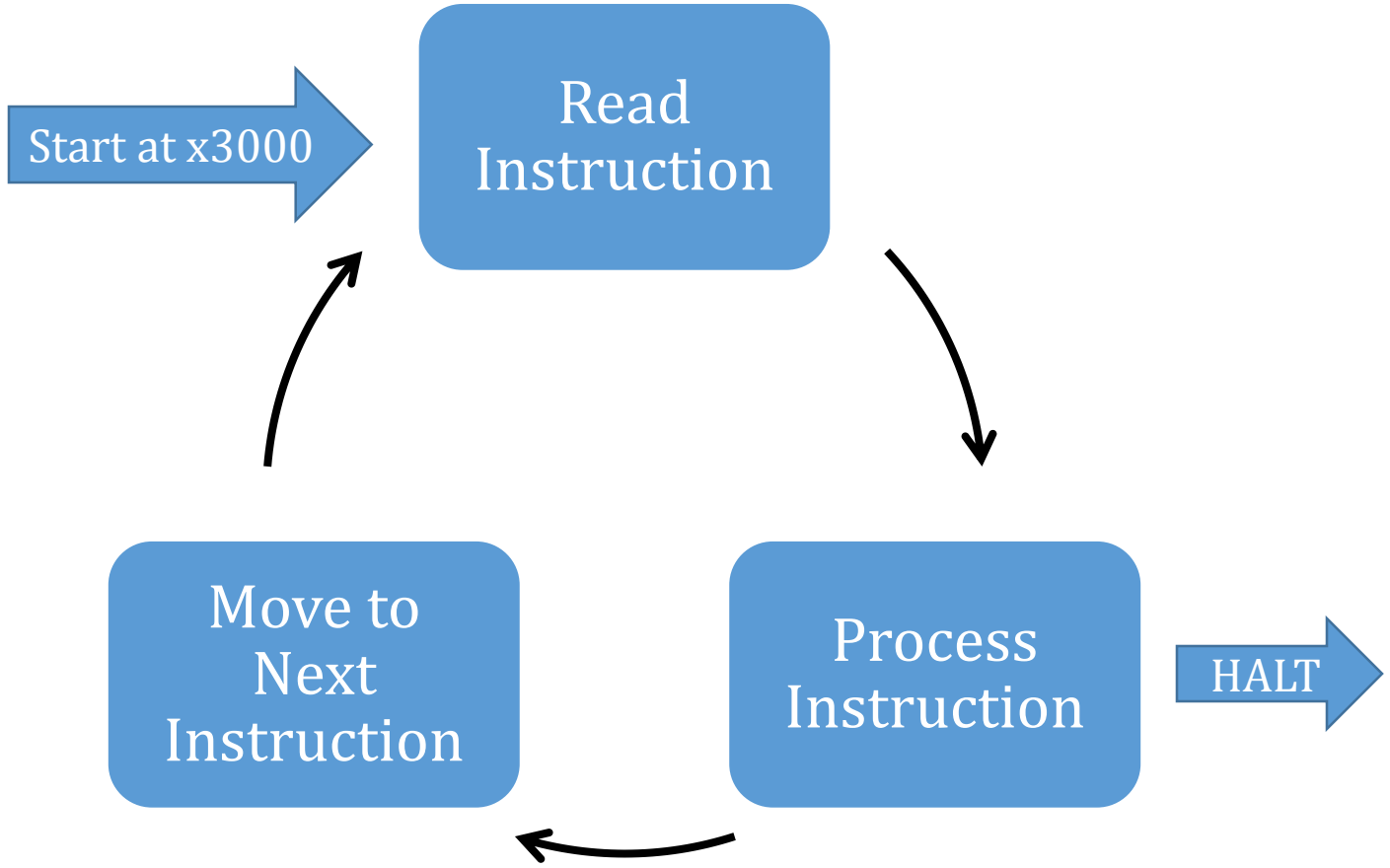
Inverts value in SR1(Ra), stores result in DR (Rz)

LC3 Boilerplate

- Start a program with: `.orig x3000`
  - Defines where to put the program in memory… more to come

- End a program with:
```
HALT
.end
```
  - HALT is an instruction which tells OS not to look for more instructions
  - .end tells assembler not to look for more instructions

LC3 Instruction Cycle (1st Draft)



LC3 Idioms

```
        AND     R3,R4,#0        ; Zero register 3

        ADD     R4,R3,#0        ; Copy R3 to R4

        ADD     R3,R3,#1        ; Increment Register 3

        NOT     R4,R3           ; R4 ← -R3
        ADD     R4,R4,#1

        ADD     R4,R3,R3        ; R4 ← R3 x 2
```

Example L3 Program: Specification

Write an LC3 program using AND, ADD, and NOT to put the value "100" into register 2. Use as few instructions as possible.

Example LC3 Program (6 instructions)

```
.orig x3000
AND  R1,R1,#0      ; R1=0
ADD R1,R1,#15      ; R1=0+15
ADD R1,R1,R1       ; R1=15+15=30
ADD R2,R1,R1       ; R2=30+30=60
ADD R2,R2,R1       ; R2=60+30=90
ADD R2,R2,#10      ; R2=90+10=100
HALT
.end
```

Example LC3 Program

```
.orig x3000
AND  R1,R1,#0      ; R1=0
ADD R1,R1,#10      ; R1=0+10=10
ADD R1,R1,R1       ; R1=10+10=20
ADD R2,R1,R1       ; R2=20+20=40
ADD R2,R2,R2       ; R2=40+40=80
ADD R2,R2,R1       ; R2=80+20=100
HALT
.end
```

Example LC3 Program

```
.orig x3000
AND  R2,R2,#0      ; R2=0
ADD R2,R2,#15      ; R2=15
ADD R2,R2,R2       ; R2=30
ADD R2,R2,R2       ; R2=30+30=60
ADD R2,R2,#-10     ; R2=60+-10=50
ADD R2,R2,R2       ; R2=50+50=100
HALT
.end
```

Example LC3 Program – 5 instructions!

```
.orig x3000
AND  R2,R2,#0      ; R2=0
ADD R2,R2,#15      ; R2=15
ADD R2,R2,#10      ; R2=25
ADD R2,R2,R2       ; R2=50
ADD R2,R2,R2       ; R2=100
HALT
.end
```