Instruction Examples + CPU Organization

Outline

- LC-3: simple computer, limited operations; e.g., integers only (“Little Computer 3”)
- Example: LC-3 Instruction Set
  - Arithmetic/logical (operate): process/compute information
  - Memory: move data between registers/memory (or I/O devices)
  - Control: change the sequence of instructions that will be executed
- Overall Organization of CPU
  - Data Path
  - Control Unit
- Overview of LC-3 Data Path Design

Example: LC-3

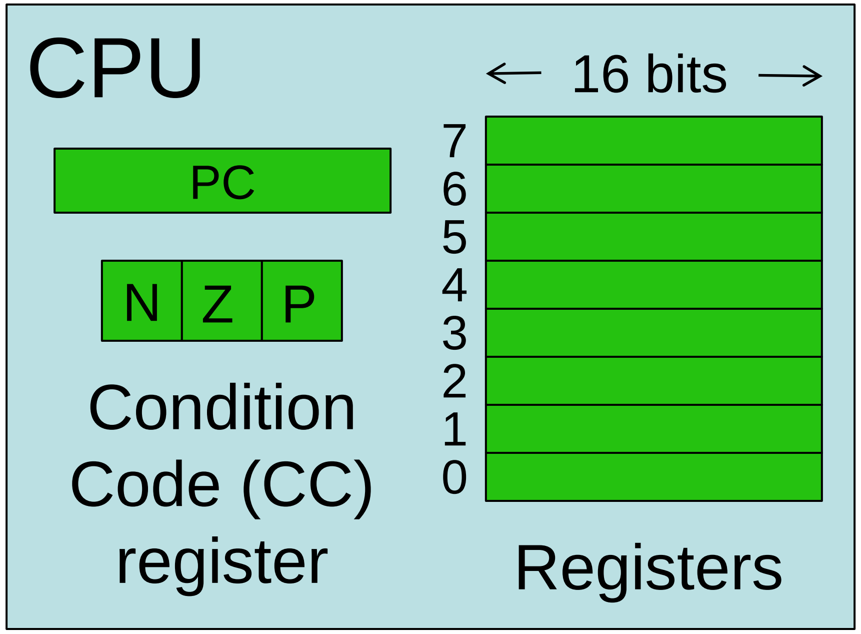
LC-3 is a 3-address, load/store RISC architecture

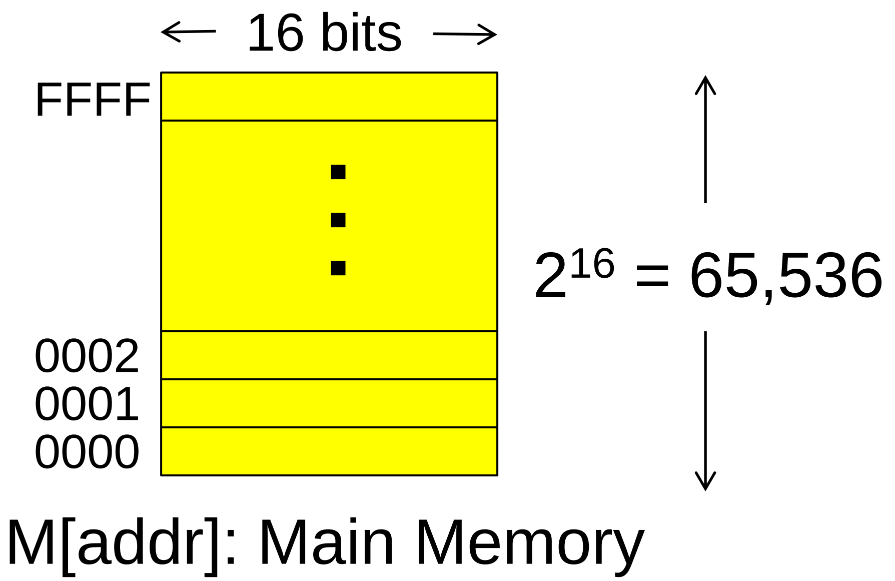
Questions Every CPU Designer Must Decide	LC-3
Size of data (bits) for "most" operations?	16-bit word
What does a memory address refer to (e.g., byte)?	16 bits (word)
How many bits in an address?	16 bits
Other storage visible to machine instructions?	8 16-bit registers; CC
What representation is used for integers?	Twos complement
What operations? Addressing modes?	LC-3 instruction set

3-address: 3 operands
Load/store: only load, store ops access memory
RISC: small # operations

Hex is convenient:
4 hex digits vs. 16 bits

Addressing modes: mechanism for specifying operand location





LC-3 Machine Instructions

All instructions are 16 bits: strings of 0's and 1's!

Three types of instructions: Arithmetic/logical, Memory, Control

Arithmetic/logical (operate)

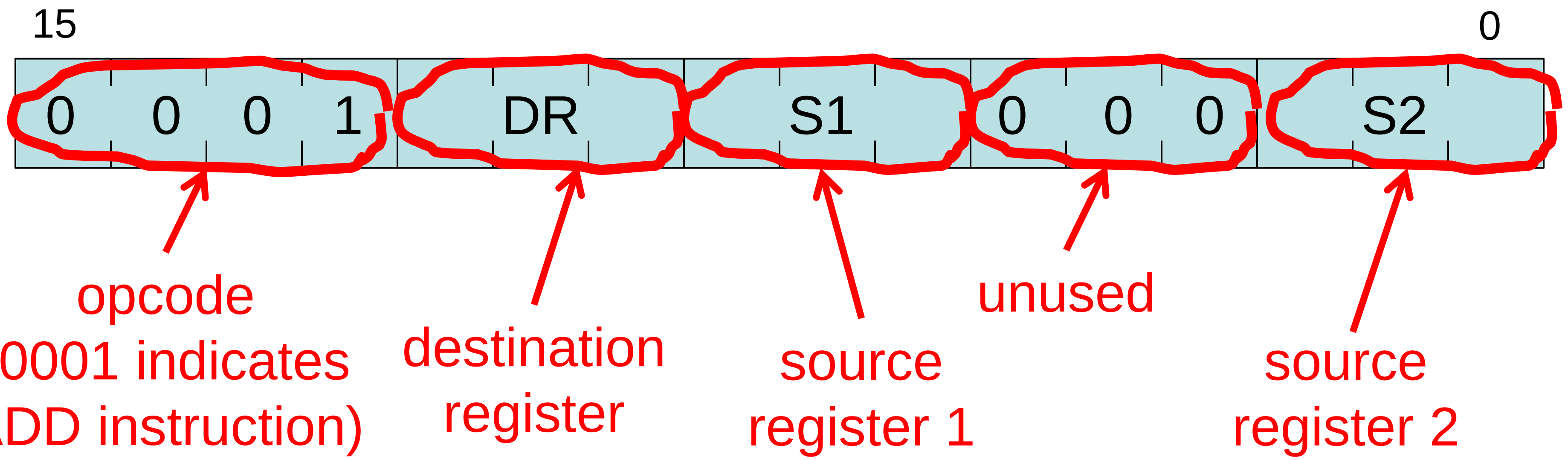
“Pseudo-assembler”
Not the same as an instruction
Human readable

```
ADD   DR, S1, S2    // DR, S1, S2 are registers
                    // DR <- S1 + S2
```

// set N, Z, P if result negative, zero, or positive, respectively

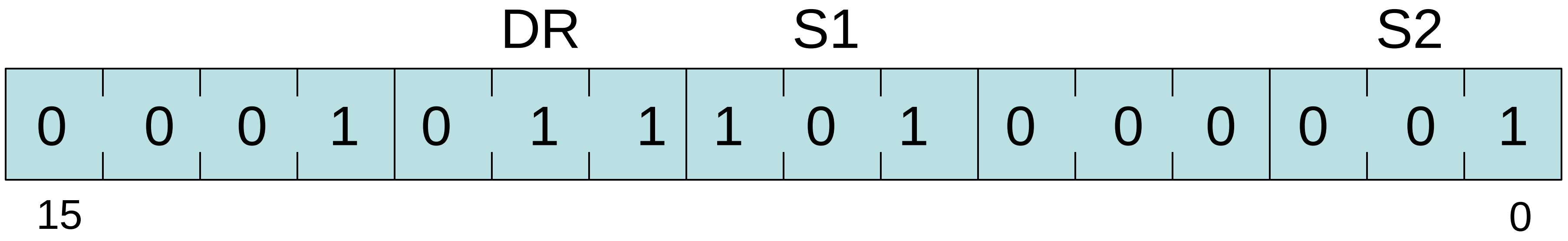
(all write instructions use; can be used to change execution sequence)

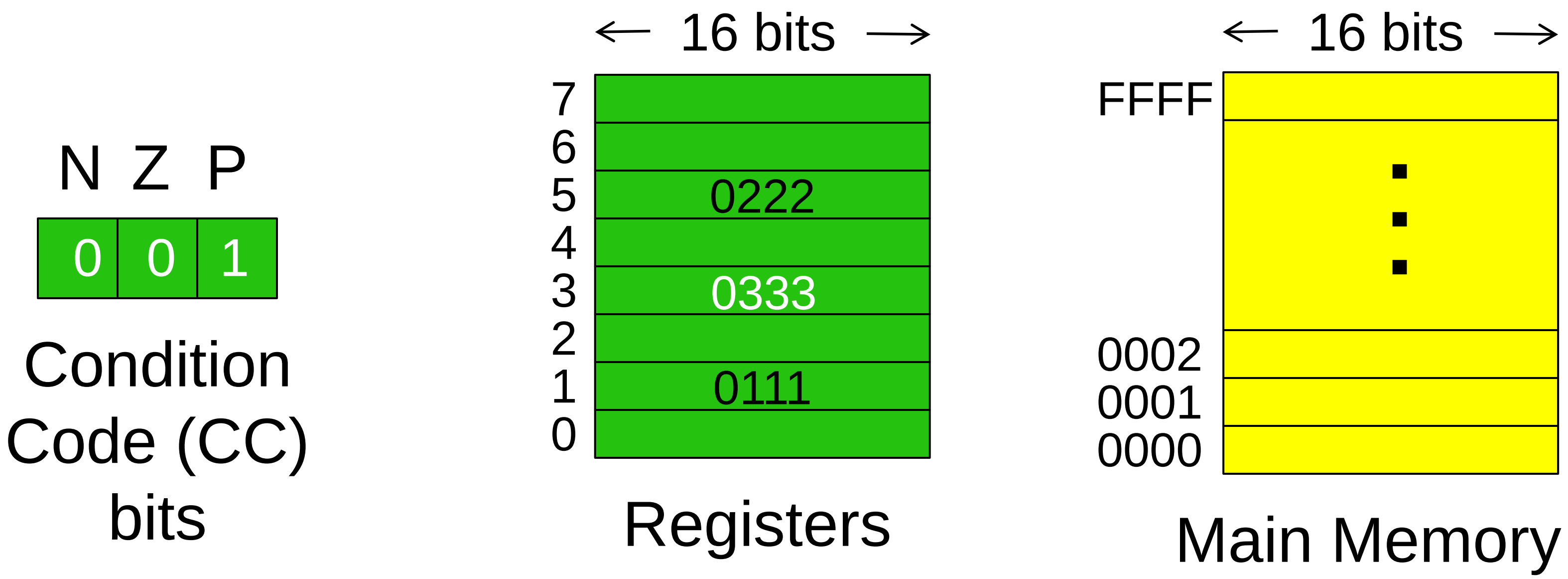
Encoding (16 bits)



LC-3 ADD Instruction

ADD R3, R5, R1 // R3 <- R5 + R1





Examples: LC-3 ADD

- What is the result of the following instruction?

					DR			S1						S2	
0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	1
15															0

← 16 bits →

7	0011
6	0030
5	0012
4	0008
3	0052
2	0030
1	0014
0	0017

Registers

- What instruction produces the change in register values shown?

← 16 bits →

7	0021
6	0051
5	0016
4	0007
3	0035
2	0012
1	0003
0	0020

Registers before

← 16 bits →

7	0021
6	0051
5	0071
4	0007
3	0035
2	0012
1	0003
0	0020

Registers after

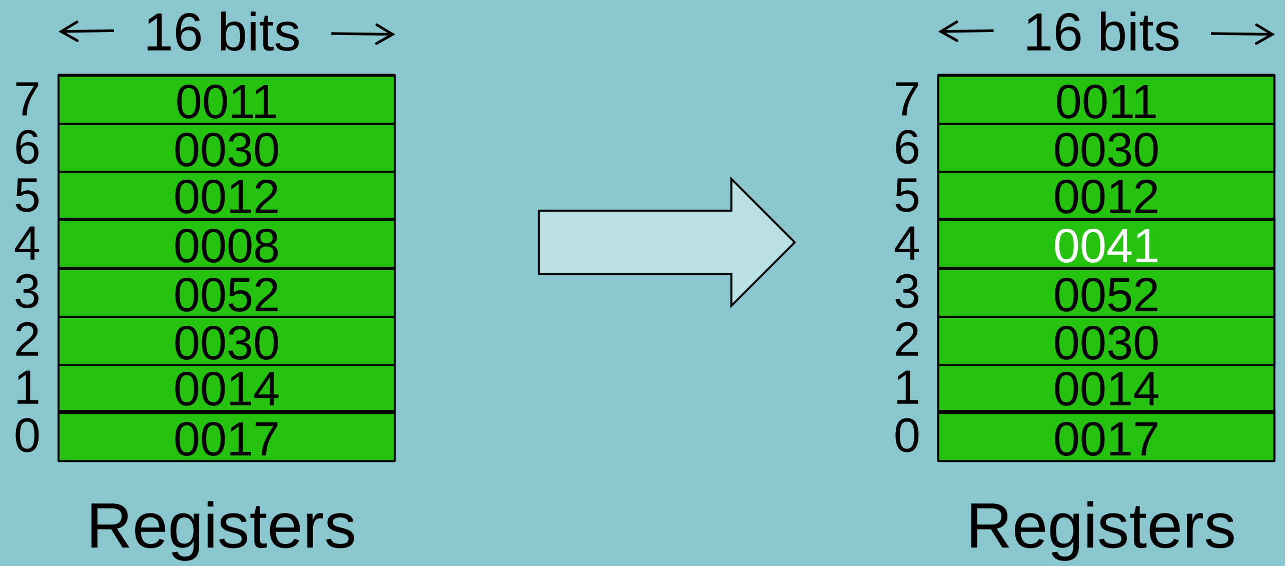
Examples: LC-3 ADD

- What is the result of the following instruction?

					DR			S1						S2	
0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	1
15															0

We add the contents of registers 010 (R2) and 111 (R7), then store the results in register 100 (R4).

0030 + 0011 = 0041



	Registers	Registers
7	0011	0011
6	0030	0030
5	0012	0012
4	0008	0041
3	0052	0052
2	0030	0030
1	0014	0014
0	0017	0017

Examples: LC-3 ADD

- What instruction produces the change in register values shown?
- Only R5 has changed so this is clearly the DR. (101) Then we look for two values that sum to 0071. Only R6 (110) and R0 (000) work.

← 16 bits →

7	0021
6	0051
5	0016
4	0007
3	0035
2	0012
1	0003
0	0020

Registers before

← 16 bits →

7	0021
6	0051
5	0071
4	0007
3	0035
2	0012
1	0003
0	0020

Registers after

- Note that two instructions give this result!

					DR			S1						S2	
0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	0

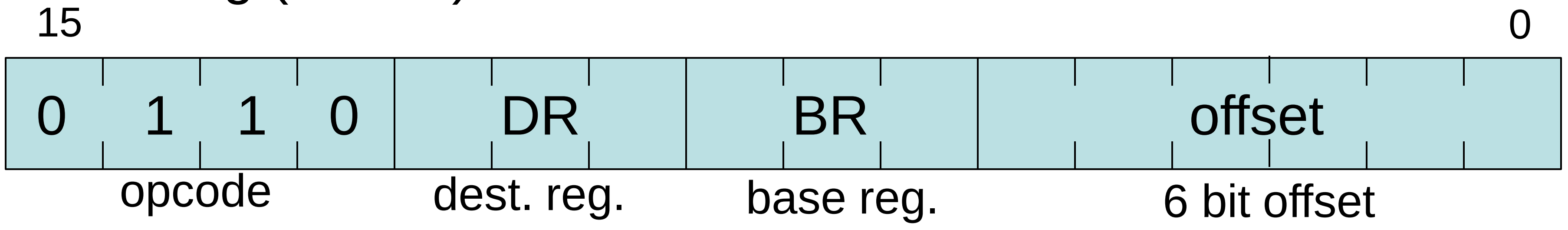
LC-3 LOAD Instruction

Memory

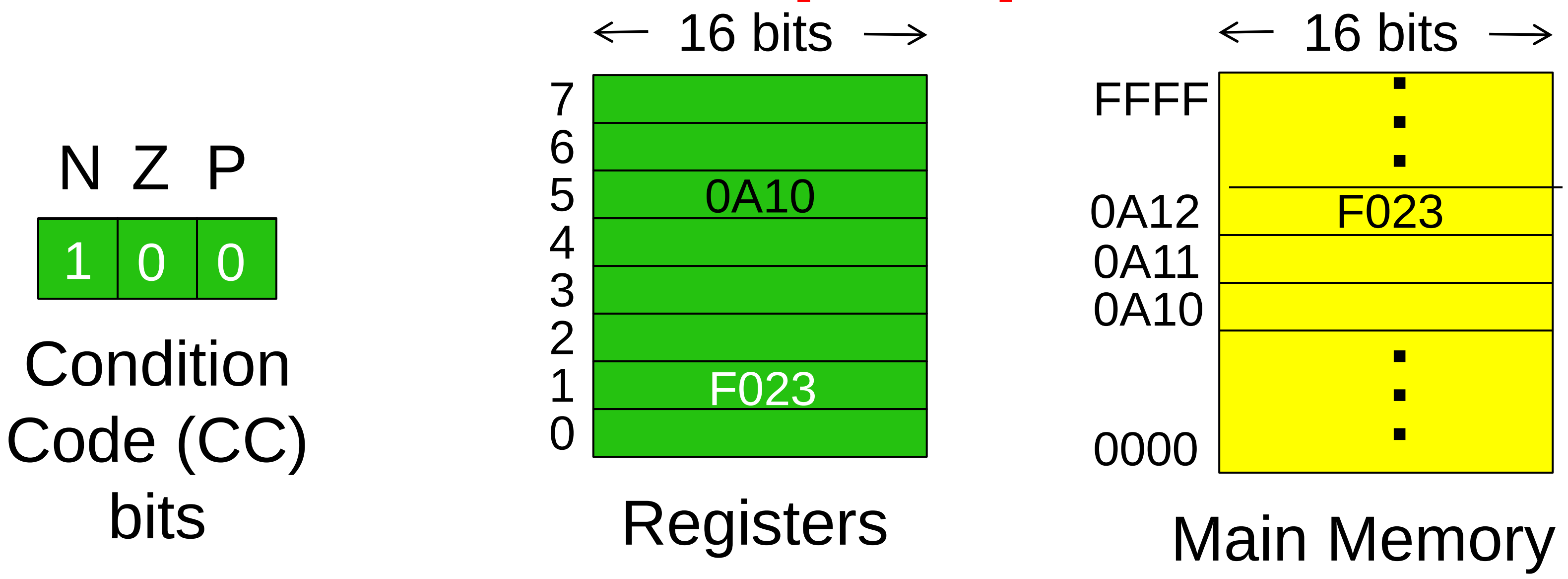
LDR DR, BR, offset // DR <- M [BR + offset]
// sign extend offset to 16 bits

Here the offset is relative to the address in BR.

Encoding (16 bits)



LDR R1, R5, #2 // R1 <- M[R5 + 2]



Why use load with base+offset mode?

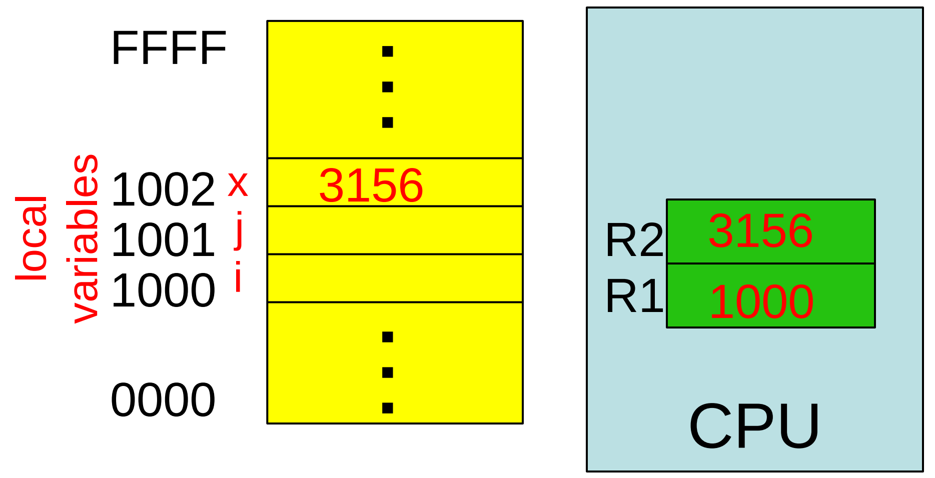
Usage Example: Local Variables

Index Addressing Mode: memory address computed by adding an offset (stored in instruction) to a base register (BR)

LDR DR, BR, offset // DR <- M [BR + offset]

C Code

```
foo ()
{ int i, j;
  int* x;
... x ...
}
```



Assembler

BR: address of memory block used for local variables

Offset: specifies variable (i:0; j:1; x:2)

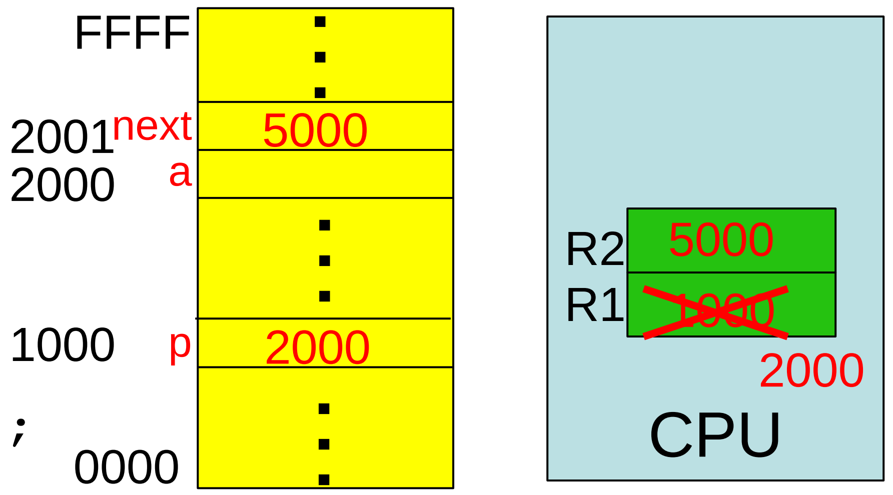
```
// assume R1 holds address of memory block for locals
LDR  R2,R1,#2     // R2<-x
```

Usage Example: Structures

LDR DR, BR, offset // DR <- M [BR + offset]

C Code:

```
struct node {
  int a;
  struct node * next;
}
struct node * p;
p=(struct node *)
    malloc(sizeof(struct node));

... p->next ...
```



Assembler

BR: address of structure

Offset: specifies field (a:0; next:1)

```
// Assume R1 holds address of p
LDR R1,R1,#0      // R1<-p
LDR R2,R1,#1      // R2<-p->next
```

Usage Example: Arrays

LDR DR, BR, offset // DR <- M [BR + offset]

C Code

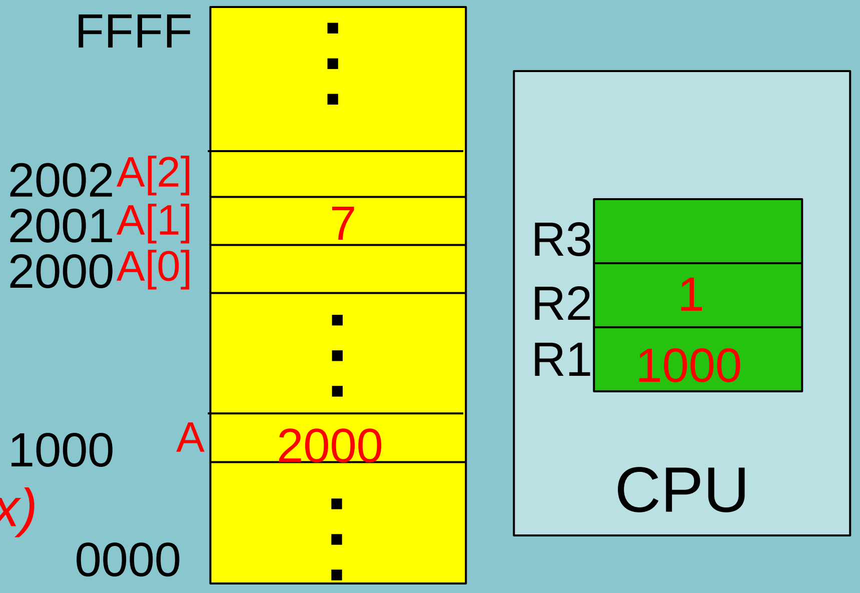
```
int *A;     // A is a pointer!
A=(int *)malloc(3*sizeof(int));

... A[i] ...
```

Assembler

BR: compute address of data

Offset: 0 *(offset from BR depends on index)*

```
// Assume R1 holds address of A
// Assume R2 holds value of i
// Compute address of A[i] in R1
LDR R1,R1,#0  // R1<-A (address of (A[0])
ADD R1,R1,R2  // R1<-address of A[i]
LDR R3,R1,#0  // R3<-A[i]
```



What is in R1, R2, and R3 after executing these ops consecutively?

Usage Example: Arrays

LDR DR, BR, offset // DR <- M [BR + offset]

C Code

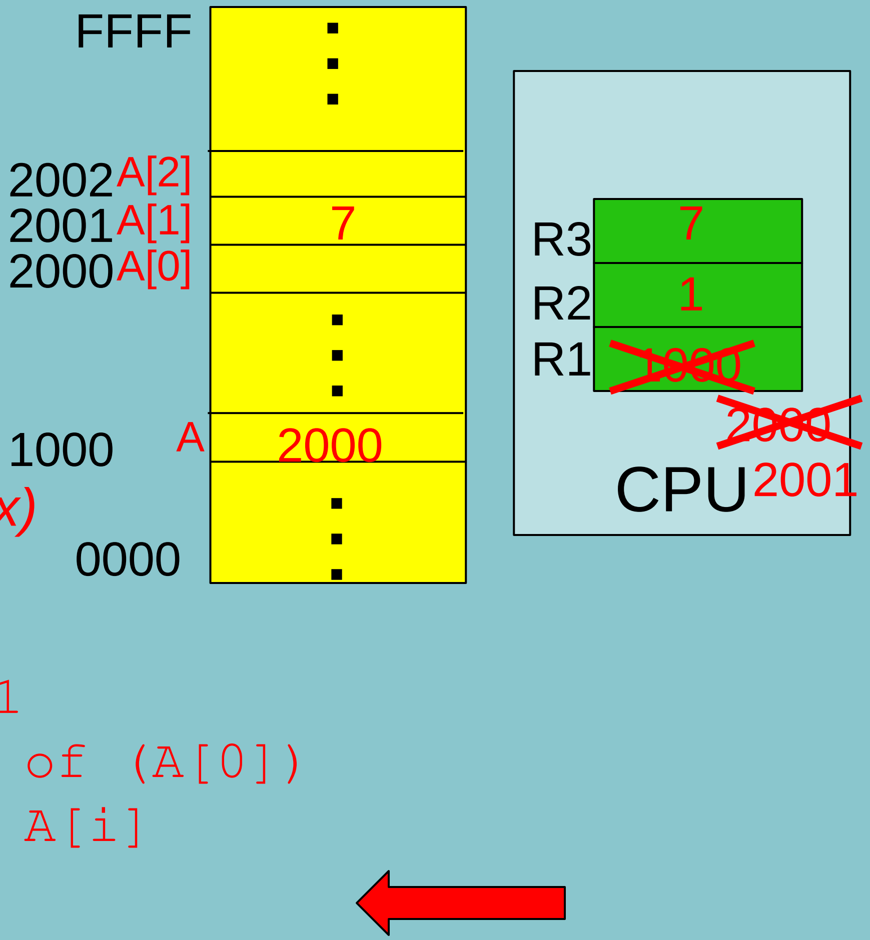
```
int *A;     // A is a pointer!
A=(int *)malloc(3*sizeof(int));

... A[i] ...
```

Assembler

BR: compute address of data

Offset: 0 *(offset from BR depends on index)*

```
// Assume R1 holds address of A
// Assume R2 holds value of i
// Compute address of A[i] in R1
LDR R1,R1,#0  // R1<-A (address of (A[0])
ADD R1,R1,R2  // R1<-address of A[i]
LDR R3,R1,#0  // R3<-A[i]
```

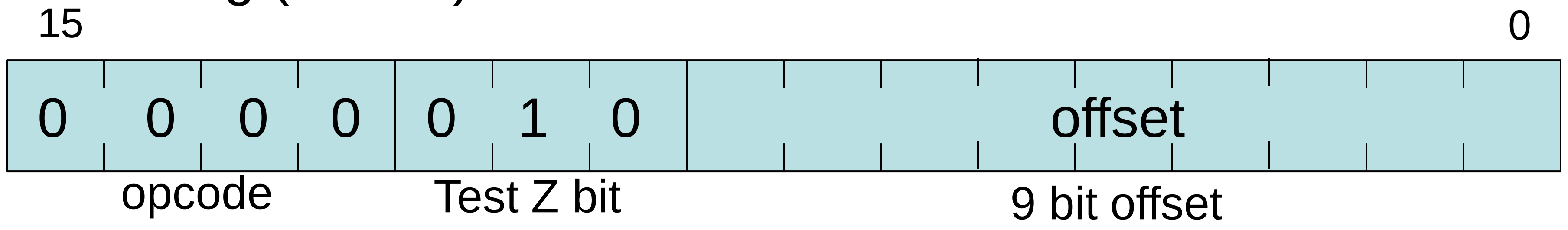


LC-3 BRANCH Instruction

Control

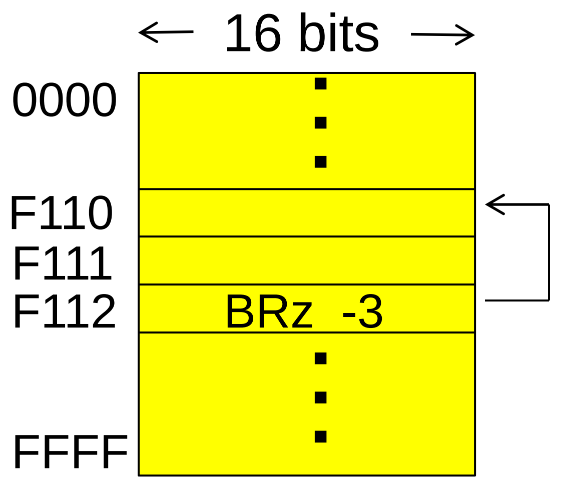
BRz offset // if (Z bit set) PC <- PC+1+offset
// sign extend offset
// previous instruction sets Z

Encoding (16 bits)



Note the offset is now relative to the (incremented) PC!

BRz -3
// if Z bit is set, PC <- PC-2 (branch taken)
// if Z not set, execute next instruction, at
// PC+1 (branch not taken)



Main Memory

Branch Instruction Usage

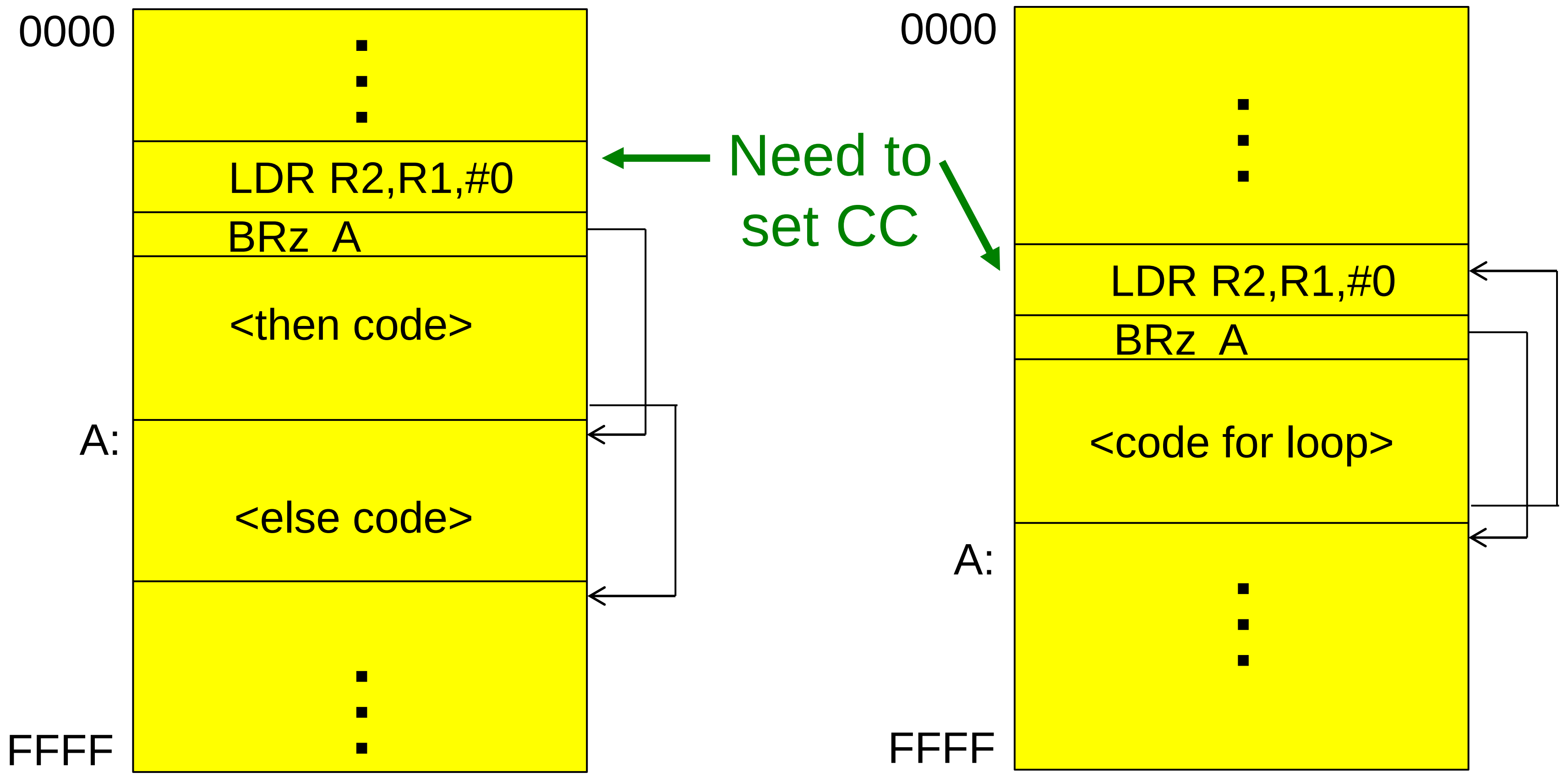
Conditional Branch

```
if (p) { then code }
else { else code }
```

Assume R1 holds address of p

Loop

```
while (p) {
    code for loop
}
```



Summary

- Machine instruction set architecture defines what aspects of the hardware are visible to the programmer/compiler
- Instruction set of LC-3 include 3 types of operations:
  - Arithmetic/logical
  - Memory
  - Control

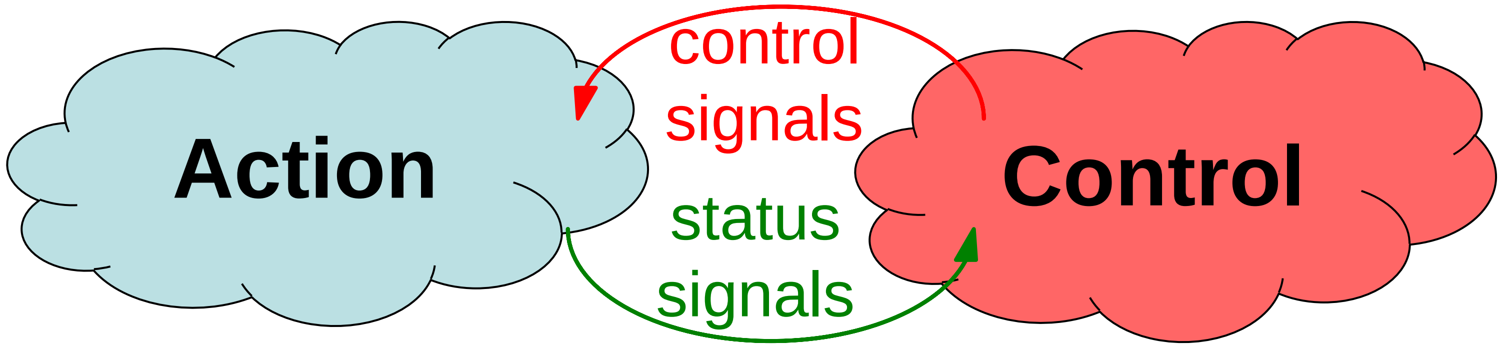
Outline

- LC-3: simple computer, limited operations; e.g., integers only
- Example: LC-3 Instruction Set
  - Arithmetic/logical (operate): process/compute information
  - Memory: move data between registers/memory (or I/O devices)
  - Control: change the sequence of instructions that will be executed
- Overall Organization of CPU
  - Data Path
  - Control Unit
- Overview of LC-3 Data Path Design

Interpreters

- An interpreter is needed to perform (execute) the instructions specified by the program
- Machine language is interpreted by CPU hardware
  - Fast, efficient execution
  - Must use a compiler to translate high level language code to machine language code
  - Examples: C, C++, FORTRAN, …
- A program (software interpreter) could also be used to interpret (execute) the program
  - Slower, less efficient execution
  - Easier (faster) to program (no compilation needed)
  - Examples: Matlab, Python, JavaScript, …

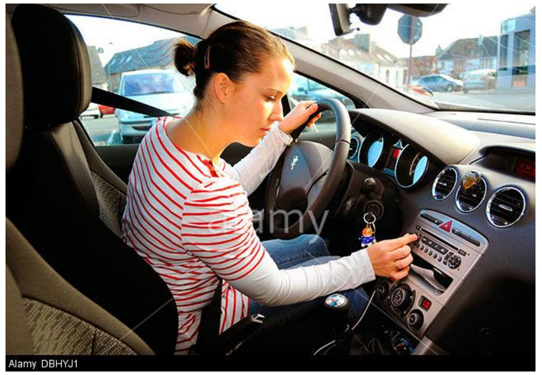
A View of Systems



Automobile

gas pedal, brake, steering wheel

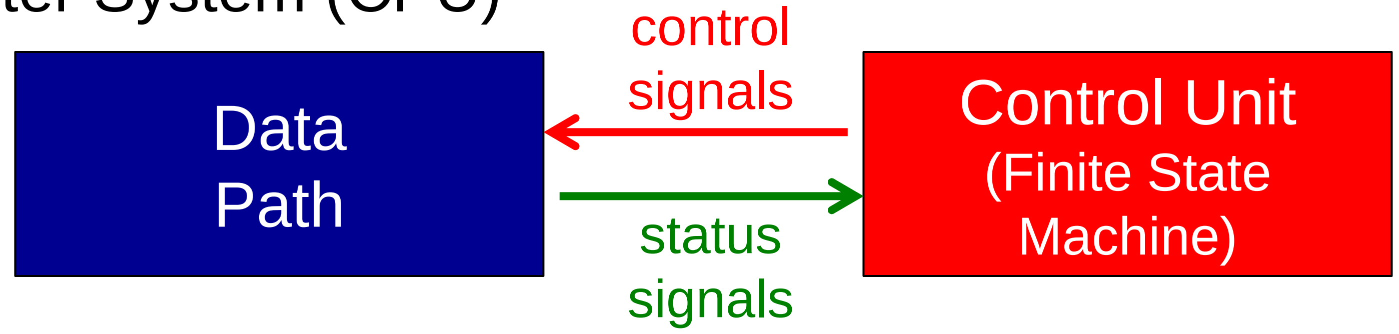
speedometer fuel gauge



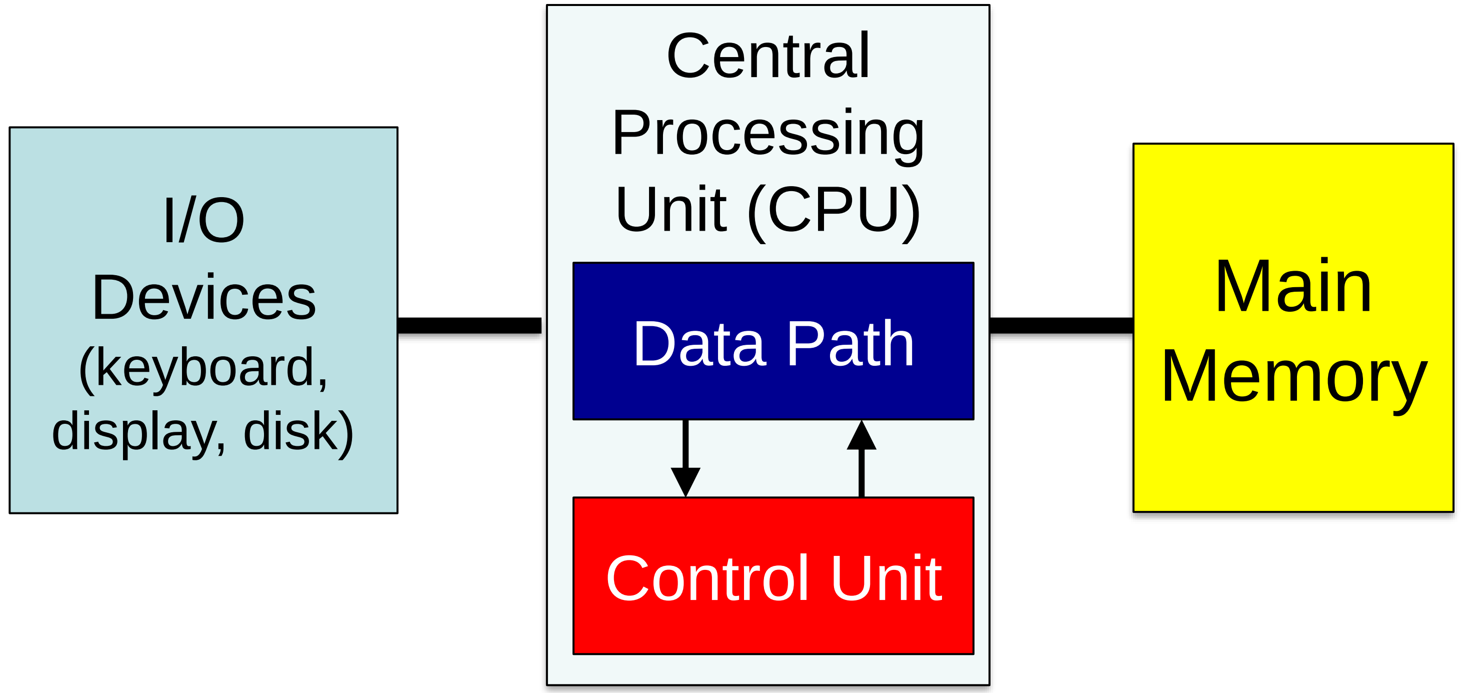
The data path includes functional units that process data (e.g., ALUs) together with registers and buses.

The control unit directs the memory, ALU, and I/O responses to instructions.

Computer System (CPU)



Von Neumann Machine Model



- The CPU basically does only one thing: move bits from one place (register or memory) to another
- Sometimes the bits are moved through a circuit that can perform arithmetic
  - Arithmetic/Logic Unit (ALU)
  - Here, ALU is purely combinational logic (no memory)

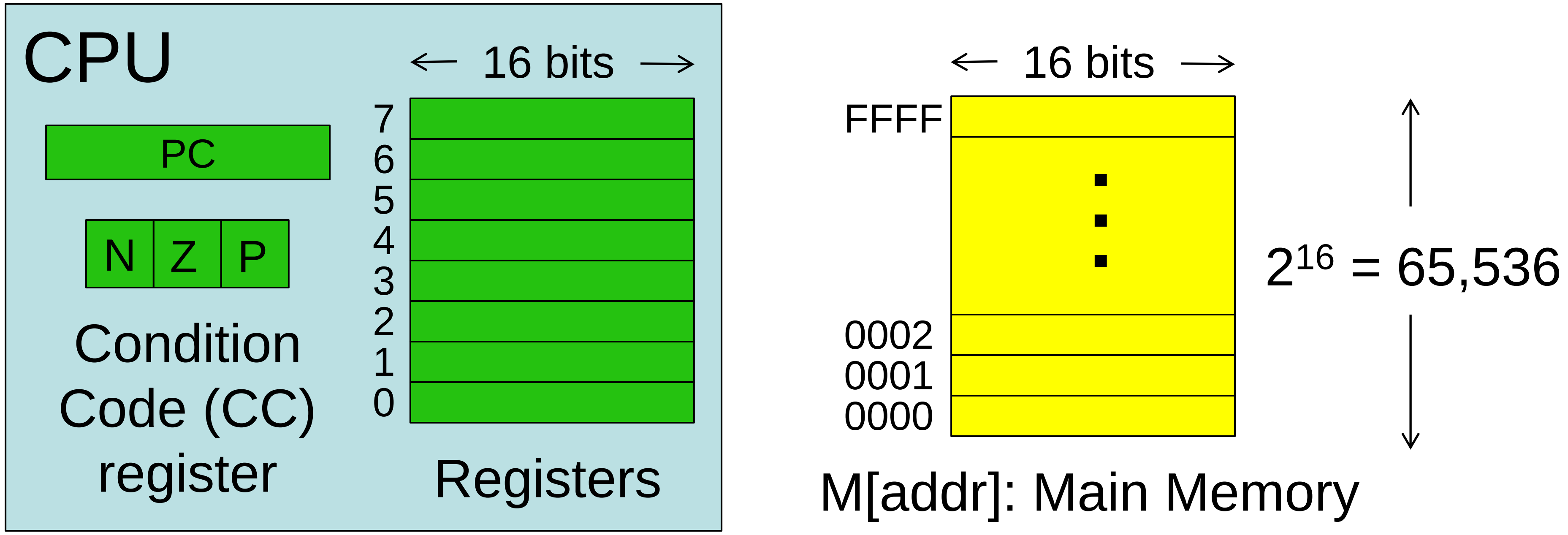
Instruction Set Architecture

- The ISA specifies all information about the computer that the software needs to be aware of
  - Memory organization
  - Register set
  - Instruction set
    - Opcodes
    - Data types
    - Addressing modes
- This information is available to a programmer/compiler when writing in the computer’s own machine language
- We will focus on the fictional LC-3 computer and right now focus on other elements of the ISA beyond instruction set

LC-3 Memory and Registers

- Address space: 2^{16} locations (16-bit addresses)
- Addressability: 16 bits (what each memory location stores)
- Normal unit of data processed in the LC-3 is 16 bits = one word, so LC-3 is word-addressable
- Accessing items from memory is typically slow, leading to the need for fast temporary storage locations that can be accessed in one clock cycle
- LC-3 has 8 general-purpose registers, each of which can store one word
- GP registers are uniquely identifiable as R0-R7

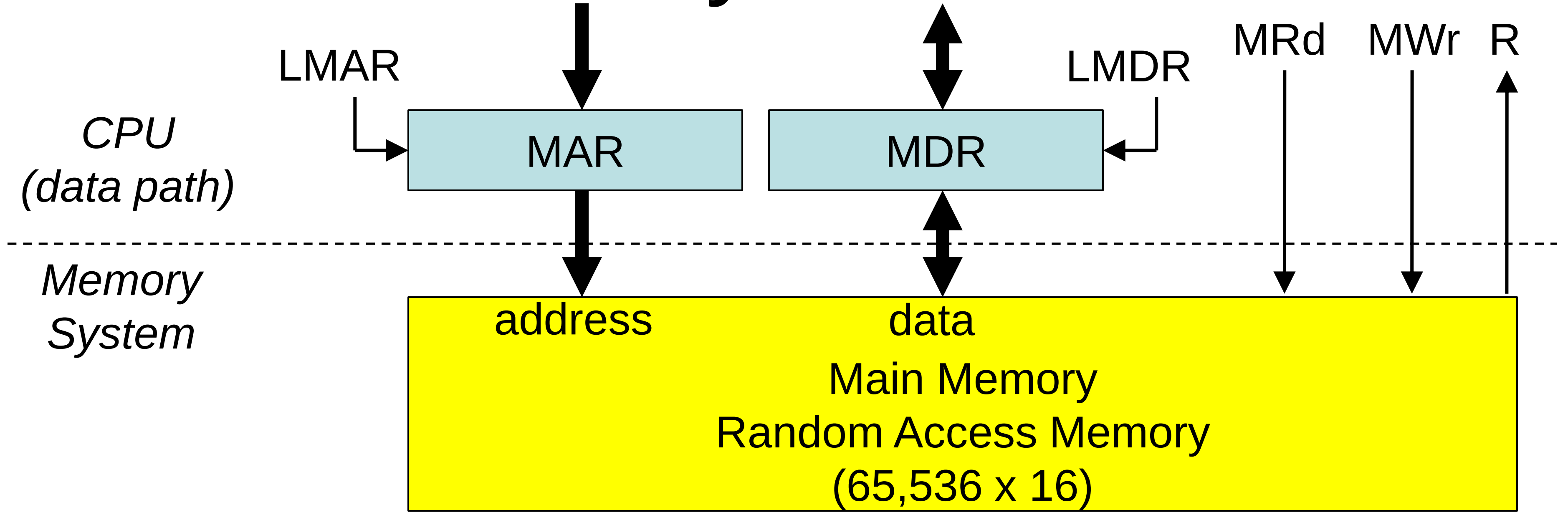
Data Path: Memory Elements



The data path consists of all the logic structures + registers and buses that combine to process information in the core of the computer.

- Program Counter (PC)
- General purpose registers
- Condition code register
- Registers are also needed to hold
  - Address used for memory reads/writes (MAR)
  - Data read from/written to main memory (MDR)
  - Instruction now being processed (IR)

Memory Interface

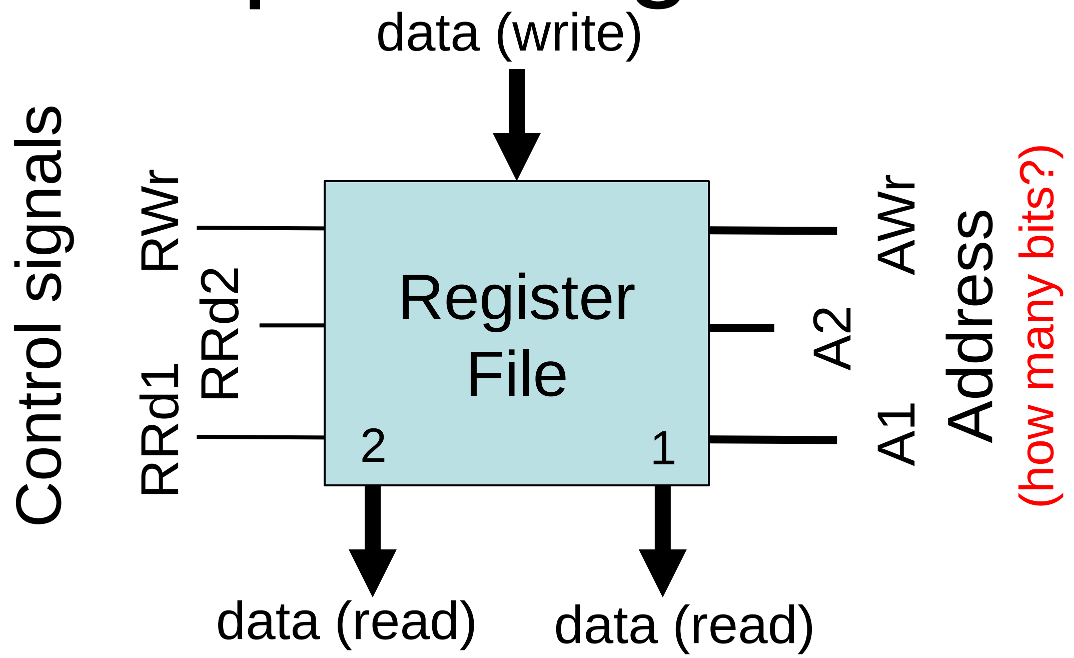


- MAR: Memory Address Register (16 bits) *read from/write to this address*
  - Control signal LMAR: assert to load MAR register
- MDR: Memory Data Register (16 bits) *information stored in that location*
  - Control signal LMDR: assert to load MDR register

Set up for read/ write ops

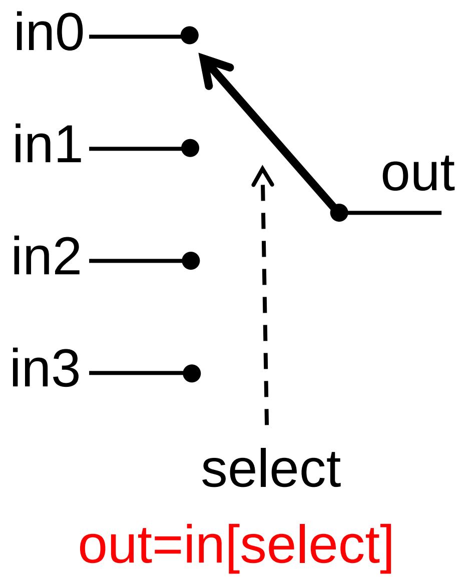
- Main Memory
  - Control signal MRd – assert to read memory: MDR<-M[MAR]
  - Control signal MWr – assert to write memory: M[MAR]<-MDR
  - Status signal R: Ready – asserted when memory operation completed

Multiport Register File

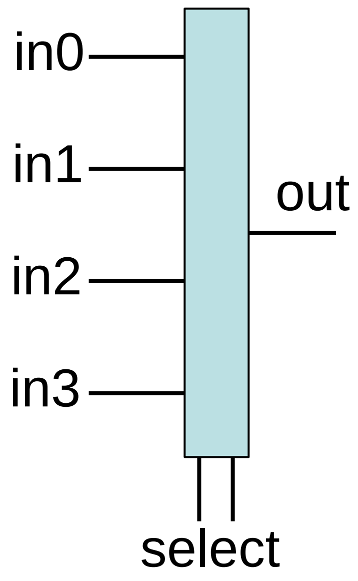


- Used to implement 8 CPU registers in LC-3
- Can simultaneously perform multiple memory operations in a single clock cycle (e.g., ADD)
  - Two read operations
  - One write operation
  - Can read/write the same register in a single instruction

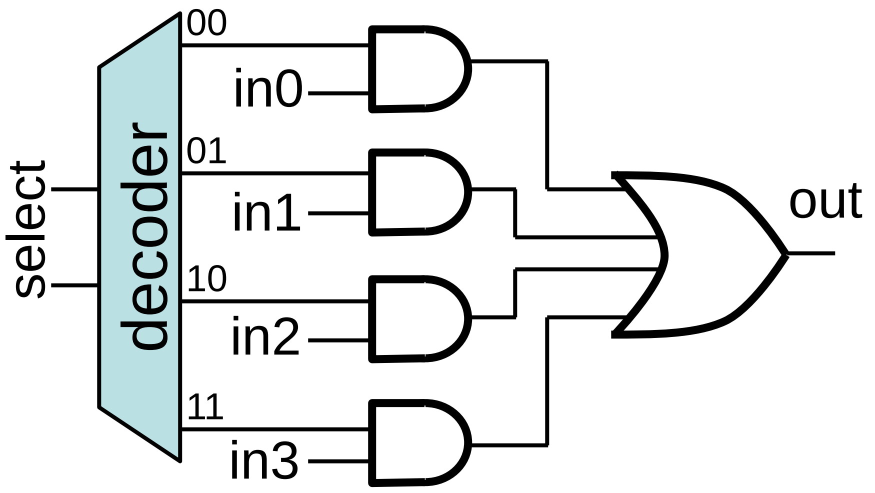
Multiplexer Circuit



logical
view



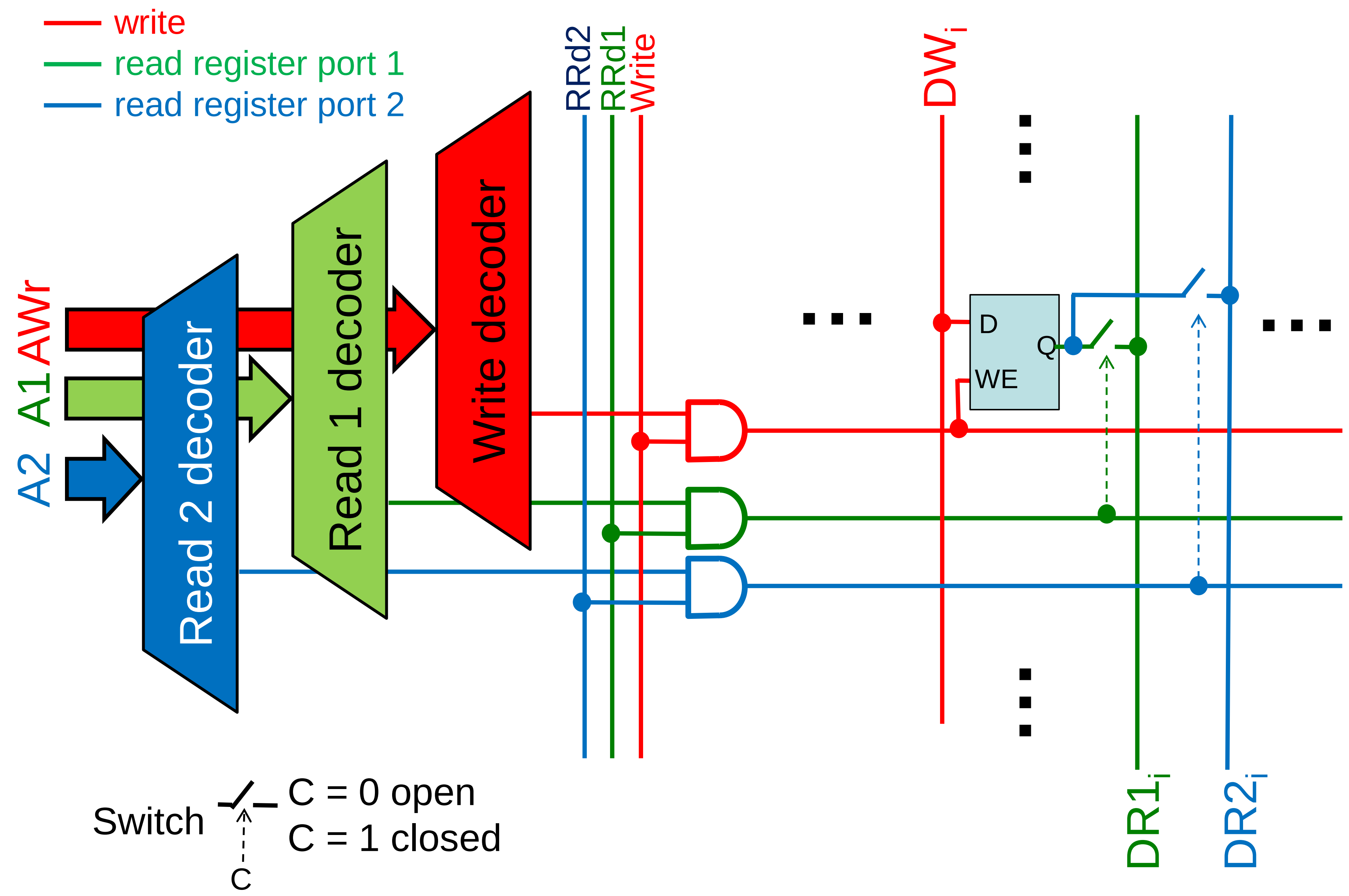
circuit
component



implementation

- Basically a way to select one input
- Combinational logic (no storage inside multiplexer)
- Above is a 4:1 one-bit multiplexer (mux)
- The circuit can be replicated for inputs with more bits

Multiport Register File Implementation



Summary

- The LC-3 is a simple computer with a simple instruction set.
- The data path consists of all the logic structures + registers and buses that combine to process information in the core of the computer.
- The control unit directs the memory, ALU, and I/O responses to instructions.
- We will see more details of the LC-3 data path next time.