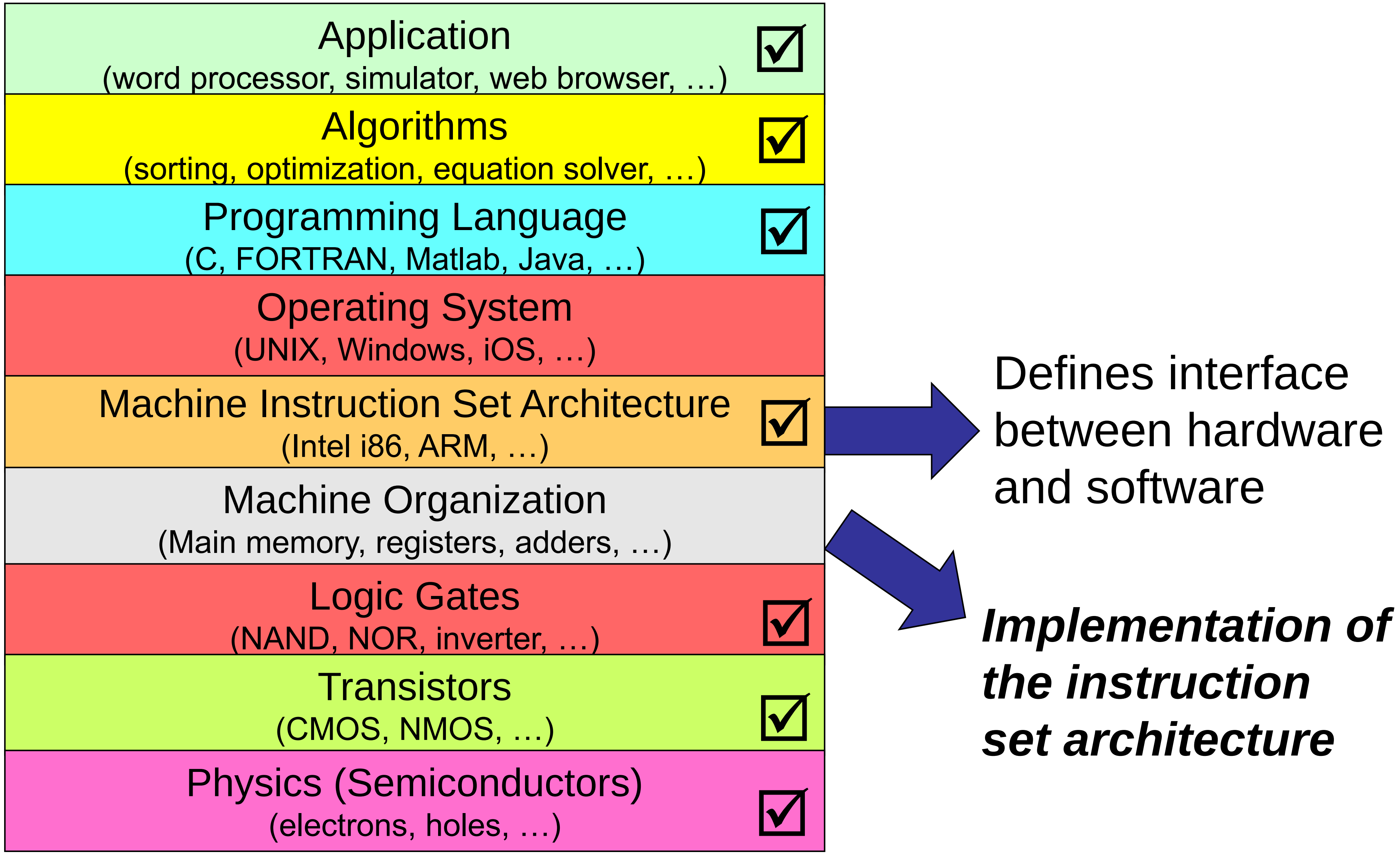
Machine Organization: Implementation of the ISA

Levels of Abstraction in Computers


Application ☑
(word processor, simulator, web browser, …)
Algorithms ☑
(sorting, optimization, equation solver, …)
Programming Language ☑
(C, FORTRAN, Matlab, Java, …)
Operating System
(UNIX, Windows, iOS, …)
Machine Instruction Set Architecture ☑
(Intel i86, ARM, …)
Machine Organization
(Main memory, registers, adders, …)
Logic Gates ☑
(NAND, NOR, inverter, …)
Transistors ☑
(CMOS, NMOS, …)
Physics (Semiconductors) ☑
(electrons, holes, …)
Defines interface between hardware and software
Implementation of the instruction set architecture

Outline

- More on LC-3 Instruction Set: Data Path Examples
  - Arithmetic/logical
  - Memory
  - Control
- Control Unit Design
  - Finite state machine

LC-3 Instruction Set

- Instruction set is defined by its set of opcodes, data types, and addressing modes (determine where the operands are located)
- The choice of which instructions to include or leave out of a new ISA depends on many factors
- Opcodes specify the types of operations; operands specify on what data the operation is performed
- LC-3 ISA has 15 instructions, each with a unique opcode specified with the first 4 bits of an instruction
- LC-3 only supports integers (no other data type)

Three Types of Instructions

- Three types of instructions: operate (arithmetic/ logical), data movement, and control
- Categorize the following LC-3 operations as best you can:

ADD
AND
BR (branch)
JMP (jump)
JSR (jump to subroutine)
LD (load)
NOT
RET (return from subroutine)
ST (store)
TRAP (system call)

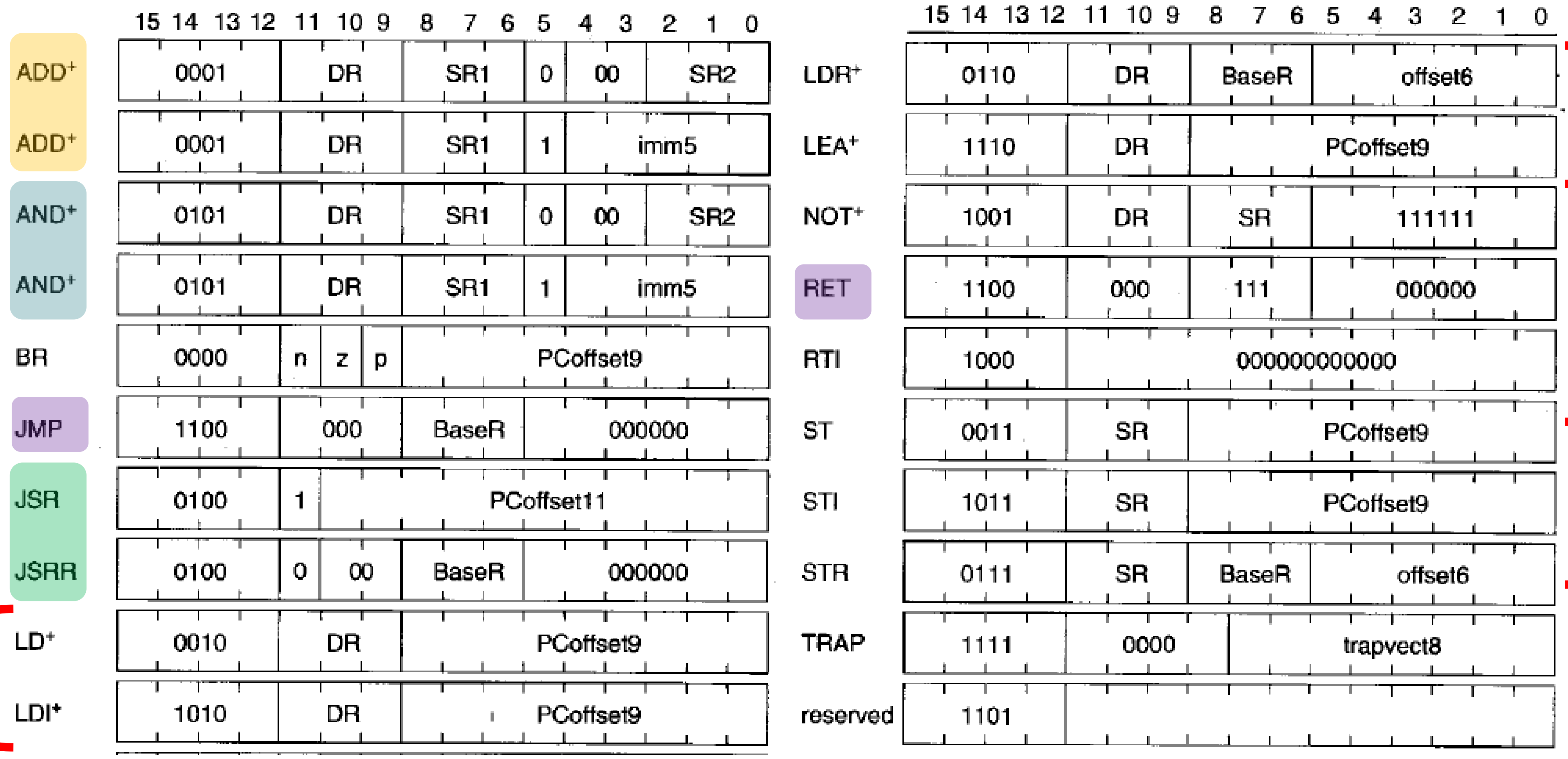
Three Types of Instructions

- Three types of instructions: operate (arithmetic/ logical), data movement, and control
- Categorize the following LC-3 operations as best you can:

Operate	**Data Movement**	**Control**
ADD	LD (load)	BR (branch)
AND	ST (store)	JMP (jump)
NOT		JSR (jump to subroutine)
		RET (return from subroutine)
		TRAP (system call)

- But wait! You may be thinking—there are not 15 of these!
- There are variations for addressing modes… next topic

LC-3 Instructions



	15 14 13 12	11	10	9	8 7 6	5	4 3	2 1 0
ADD+	0001	DR			SR1	0	00	SR2
ADD+	0001	DR			SR1	1	imm5	
AND+	0101	DR			SR1	0	00	SR2
AND+	0101	DR			SR1	1	imm5	
BR	0000	n	z	p	PCoffset9			
JMP	1100	000			BaseR	000000		
JSR	0100	1	PCoffset11					
JSRR	0100	0	00		BaseR	000000		
LD+	0010	DR			PCoffset9			
LDI+	1010	DR			PCoffset9			

	15 14 13 12	11 10 9	8	7 6	5	4 3 2 1 0
LDR+	0110	DR	BaseR		offset6	
LEA+	1110	DR	PCoffset9			
NOT+	1001	DR	SR		111111	
RET	1100	000	111		000000	
RTI	1000	000000000000				
ST	0011	SR	PCoffset9			
STI	1011	SR	PCoffset9			
STR	0111	SR	BaseR		offset6	
TRAP	1111	0000		trapvect8		
reserved	1101					

Load “flavors” (LDR+, LEA+)

Store “flavors” (ST, STI, STR)

Load “flavors” (LD+, LDI+)

- But wait! You may be thinking—there are more than 15 of these!
- There are 15 distinct opcodes, but a few can be used in multiple ways.

Addressing Modes

- Mechanism for specifying where the operand is located, generally one of the following:
  - In memory
  - In a register
  - Within the instruction (literal or immediate operand)
- LC-3 supports 5 addressing modes:
  - Immediate: operand is in the instruction
  - Register: operand is in a register
  - PC-relative: operand is located an offset from the PC
  - Indirect: PC-relative address is a pointer to the operand
  - Base + offset: operand is located an offset from address in register
- Operate instructions use only immediate and register modes (load/store architecture)

Condition Codes

- Sequence of instructions may change depending on previously generated result
- LC-3 sets or clears 3 single-bit registers each time one of the GP registers is written (arithmetic/logical and load instructions): N, Z, P
- The condition of that bit can be used by control instructions to change the sequence of instructions

Instructions to Be Implemented

```
ADD   DR, S1, S2          // DR <- S1 + S2
LDR   DR, BR, offset      // DR <- M [BR + offset]
BRz   offset              // if (Z bit set)
                          //      PC <- PC+1+offset
```

We will look at these in detail and look at a few others briefly

Instruction Processing

1. Fetch instruction to be executed: M[PC] and increment the PC
2. Decode instruction (check opcode)
3. Perform instruction
   - Evaluate address
   - Fetch operands
   - Execute operation (e.g., add)
   - Store result
4. Go back to step 1

LC-3 ADD Instruction

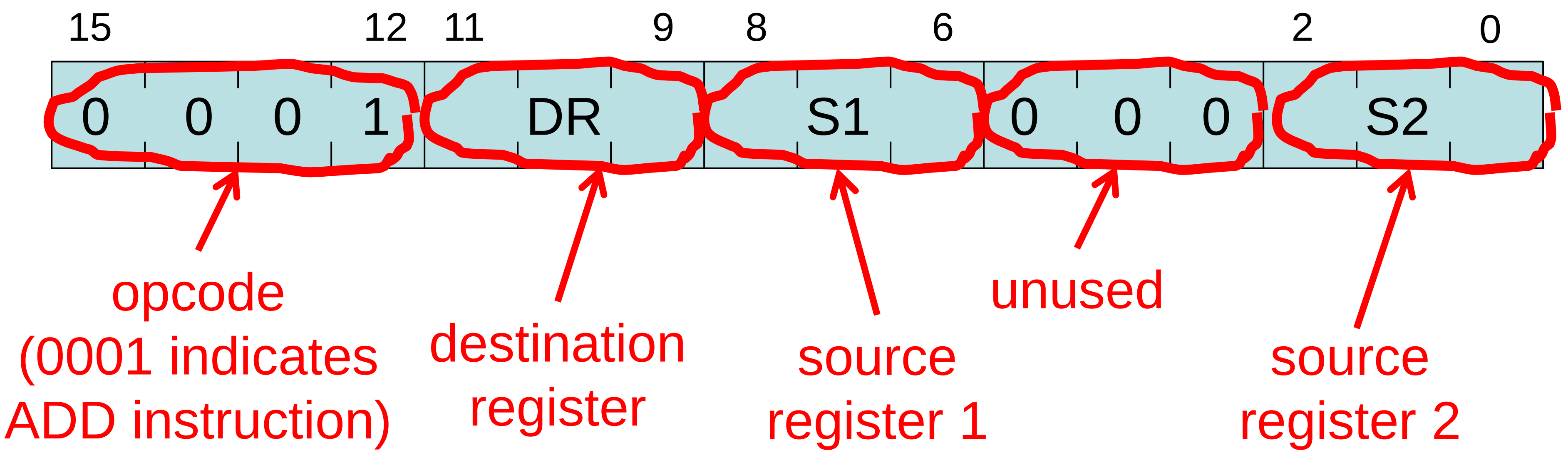
Notation: IR[x:y] means bits x, x-1, … y of IR register

IR[15:12] is the opcode; IR[8:6] indicates register S1

Arithmetic

ADD DR, S1, S2 // DR, S1, S2 are registers
// DR <- S1 + S2
// set N, Z, P if result negative, zero, or positive, respectively

Encoding (16 bits)



LC-3 ADD Instruction

Encoding (16 bits)

15			12	11		9	8		6				2		0
0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1

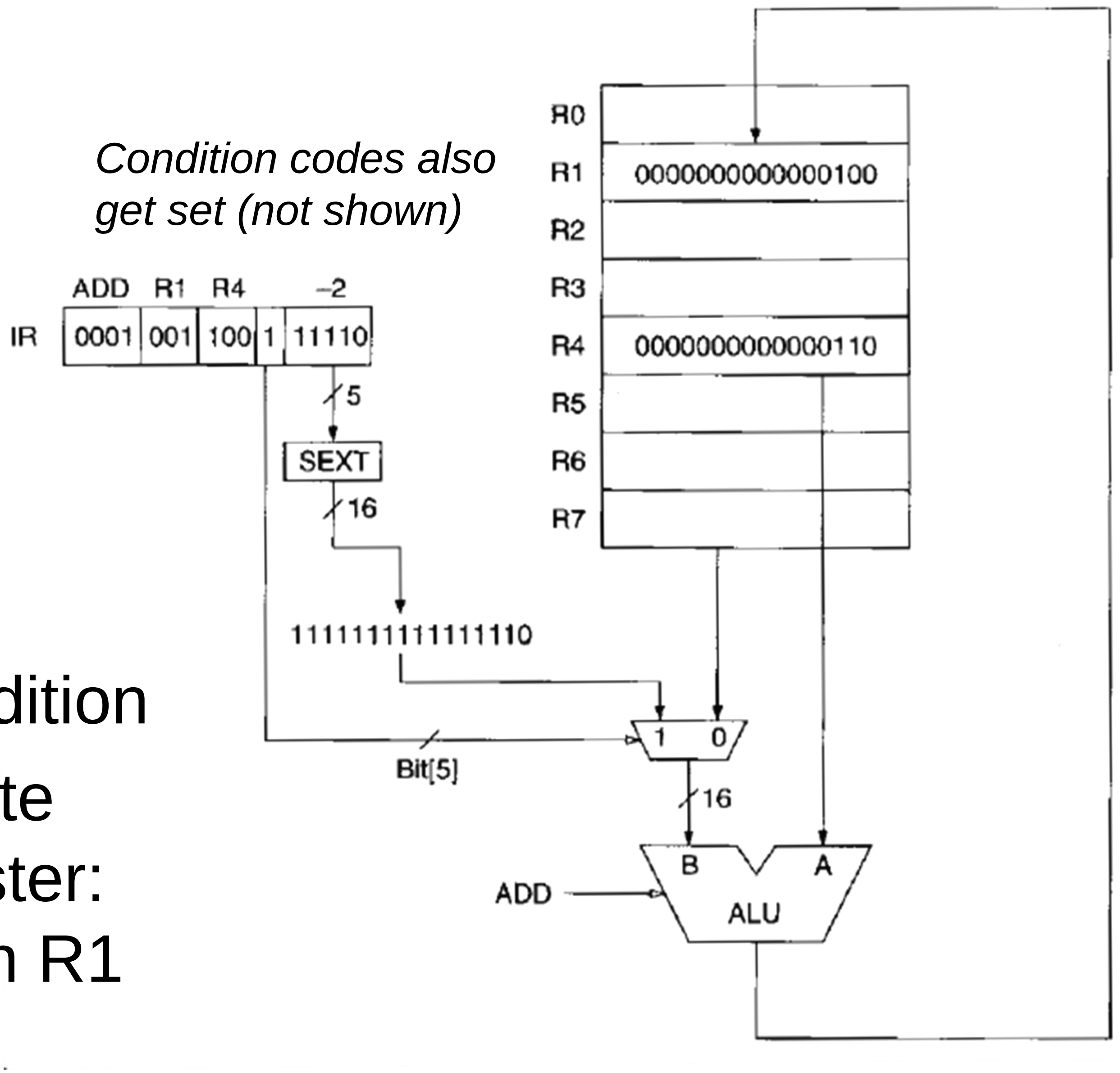
ADD R1, R4, R5

// R1 <- R4 + R5

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

- ADD performs 2s complement addition
- ADD can also specify an immediate operand instead of a second register: 0001 001 100 1 11110 will store in R1 the sum of R4 and -2

Condition codes also get set (not shown)



LC-3 Logical Instructions

- See if you can determine the effects of the following LC-3 instructions.

	15			12	11		9	8		6				2		0
NOT	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1
						DR			S1				Must contain all 1’s			

	15			12	11		9	8		6				2		0
AND	0	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0
						DR			S1			Must be 0’s			S2	

7	721C
6	382A
5	0001
4	F0F0
3	0004
2	E1E1
1	02A3
0	B782

Registers

LC-3 Logical Instructions

- See if you can determine the effects of the following LC-3 instructions.

	15			12	11		9	8		6				2		0
NOT	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1
						DR			S1				Must contain all 1’s			

Store the complement of R5 in R3: FFFE

	15			12	11		9	8		6				2		0
AND	0	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0
						DR			S1			Must be 0’s			S2	

Store the bitwise AND of R4 and R2 in R6: E0E0

7	721C
6	382A
5	0001
4	F0F0
3	0004
2	E1E1
1	02A3
0	B782

Registers

LC-3 NOT Instruction

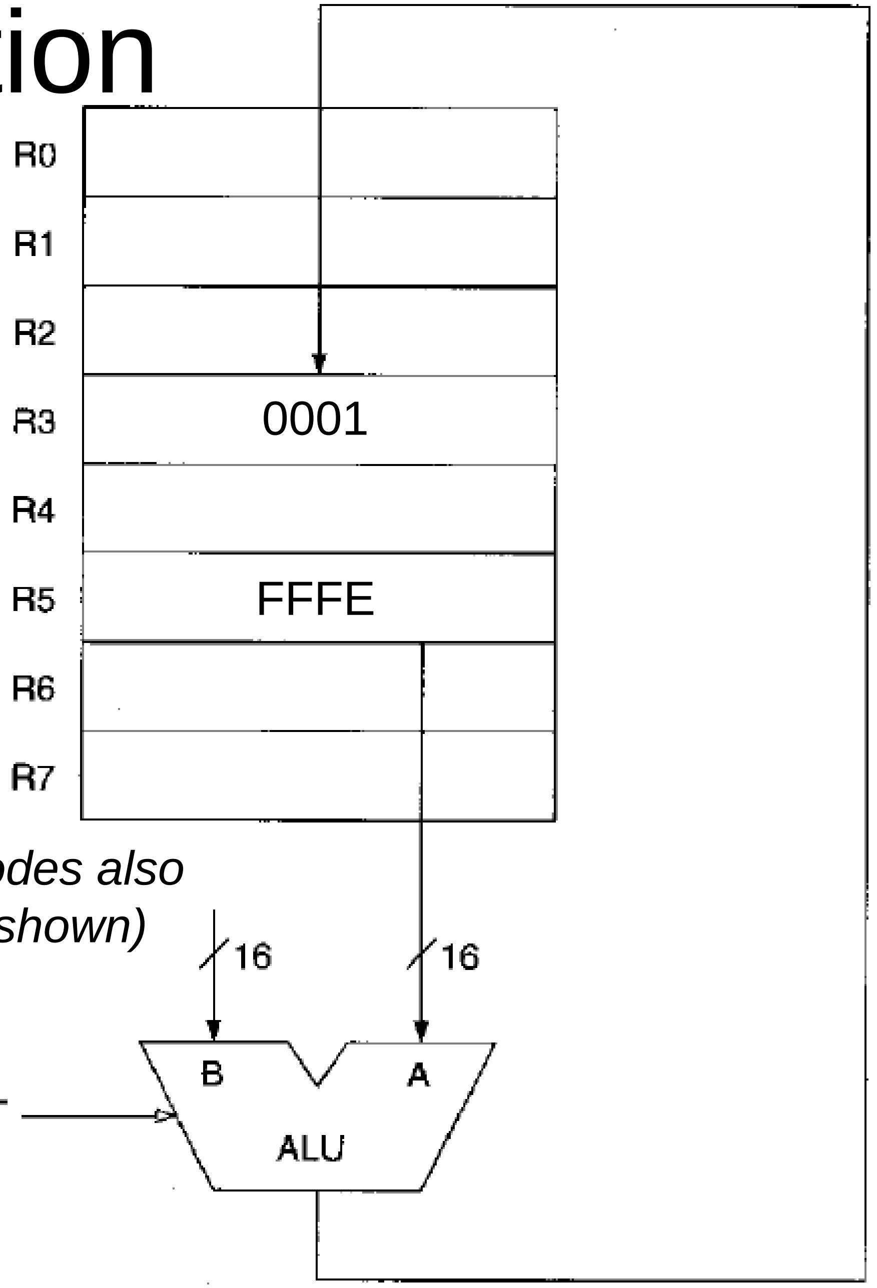
NOT

15			12	11		9	8		6				2		0
1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1
					DR			S1		Must contain all 1’s					

Store the complement of R5 in R3: FFFE

Note that only one input of the two available to the ALU is used.

Condition codes also get set (not shown)



LC-3 LDR Instruction

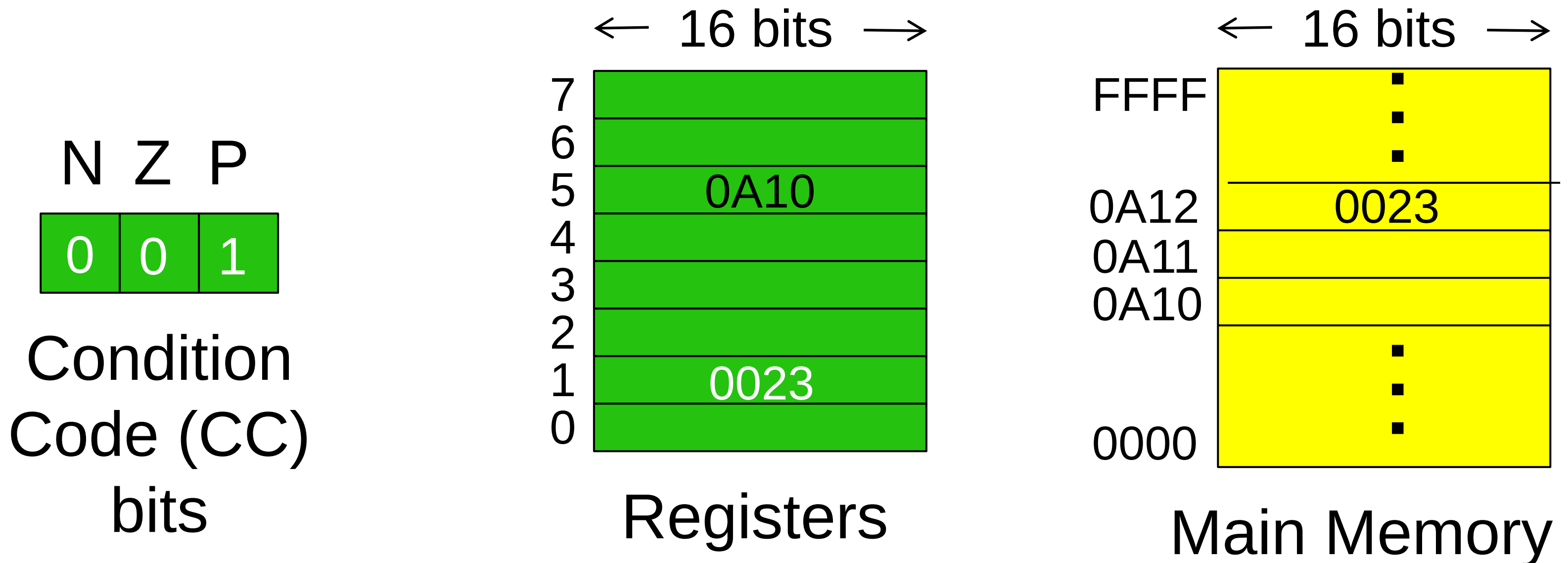
Memory

LDR RD, RS1, offset // RD <- M [RS1 + offset]
// sign extend offset

Encoding (16 bits)

15			12	11		9	8		6	5					0
0	1	1	0	RD			RS1			offset					
opcode				dest. reg.			base reg.			6 bit offset					
0	1	1	0	0	0	1	1	0	1	0	0	0	0	1	0

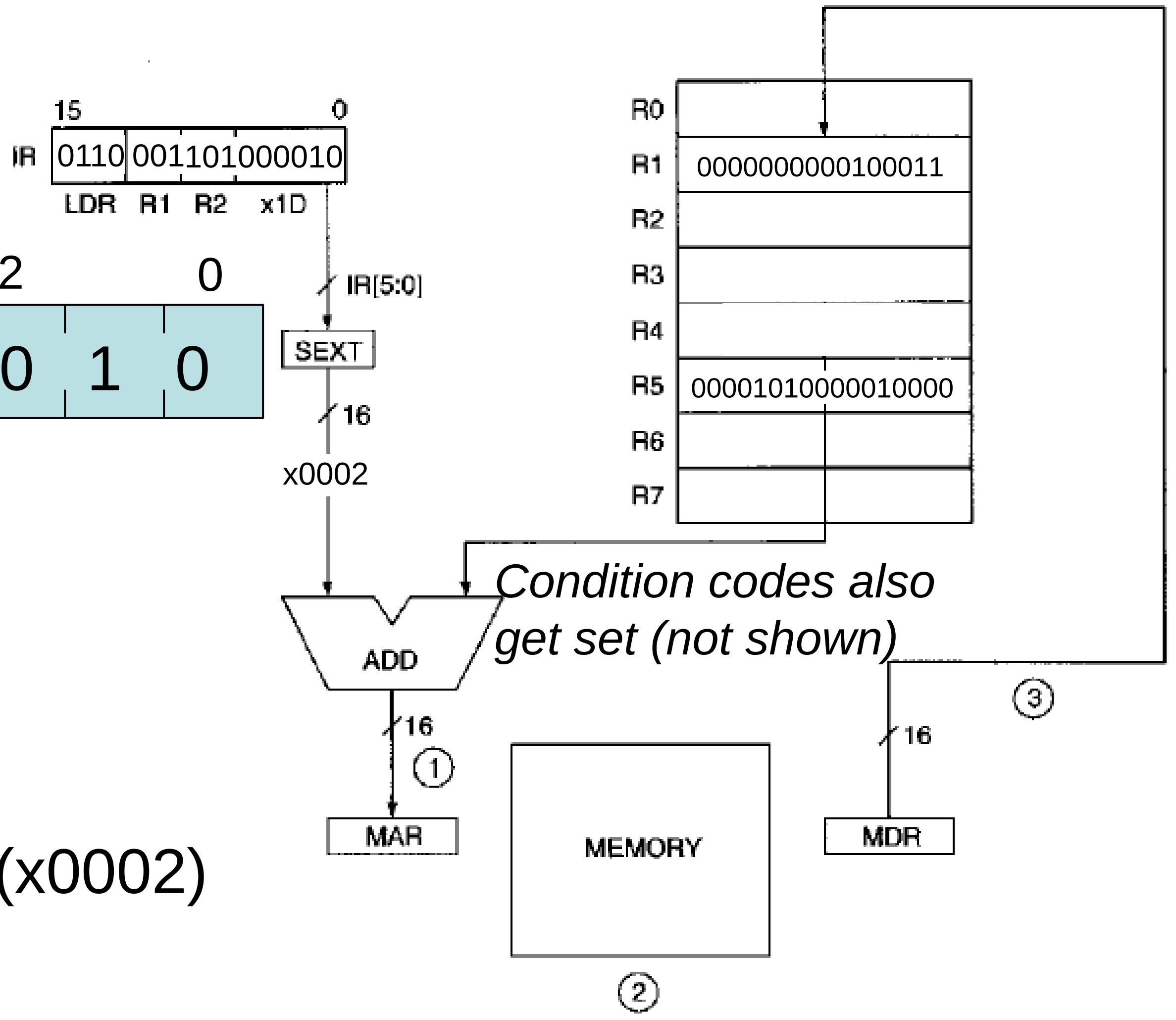
LDR R1, R5, 2 // R1 <- M[R5 + 2]



LC-3 LDR Instruction

Same LDR example: Data path

15			12	11		9	8		6				2		0
0	1	1	0	0	0	1	1	0	1	0	0	0	0	1	0
					DR			S1							



Load into R1 the contents of the memory location specified by the sum of R5 and 2: M[R5+2]

R5 contains x0A10. Adding sign-extended 2 (x0002) gives x0A12; this is loaded into the MAR.

The value at M[0A12] is loaded into the MDR and then into R1.

Other LC-3 Load Instructions

- There are a total of four load instructions in the LC-3 instruction set.
- LDR (0110):
  - Base + offset: memory address is a 6-bit offset from an address in register
- LD (0010):
  - PC-relative: memory address is located a 9-bit offset from the incremented PC
- LDI (1010):
  - Indirect: PC-relative (9-bit offset) address contains a pointer to the memory address
- LEA (load effective address) (1110):
  - Immediate: does not access memory; loads into the specified register the address formed from the incremented PC and the 9-bit value in the instruction

LC-3 BRz Instruction

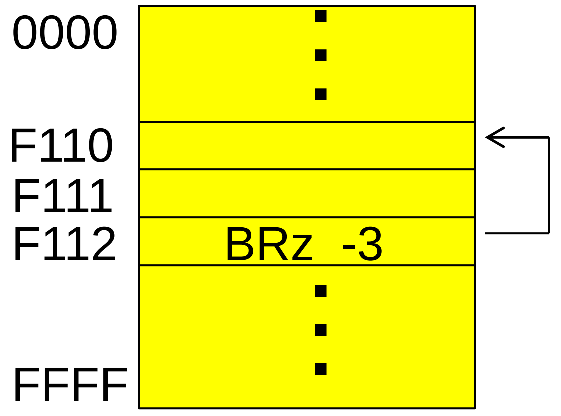
Control

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

Encoding (16 bits)

15			12	11		9	8								0
0	0	0	0	0	1	0	offset								
opcode				Test Z bit			9-bit offset								
0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	1

BRz -3
// if Z bit is set, PC <- PC-2



Main Memory

LC-3 BRz Instruction

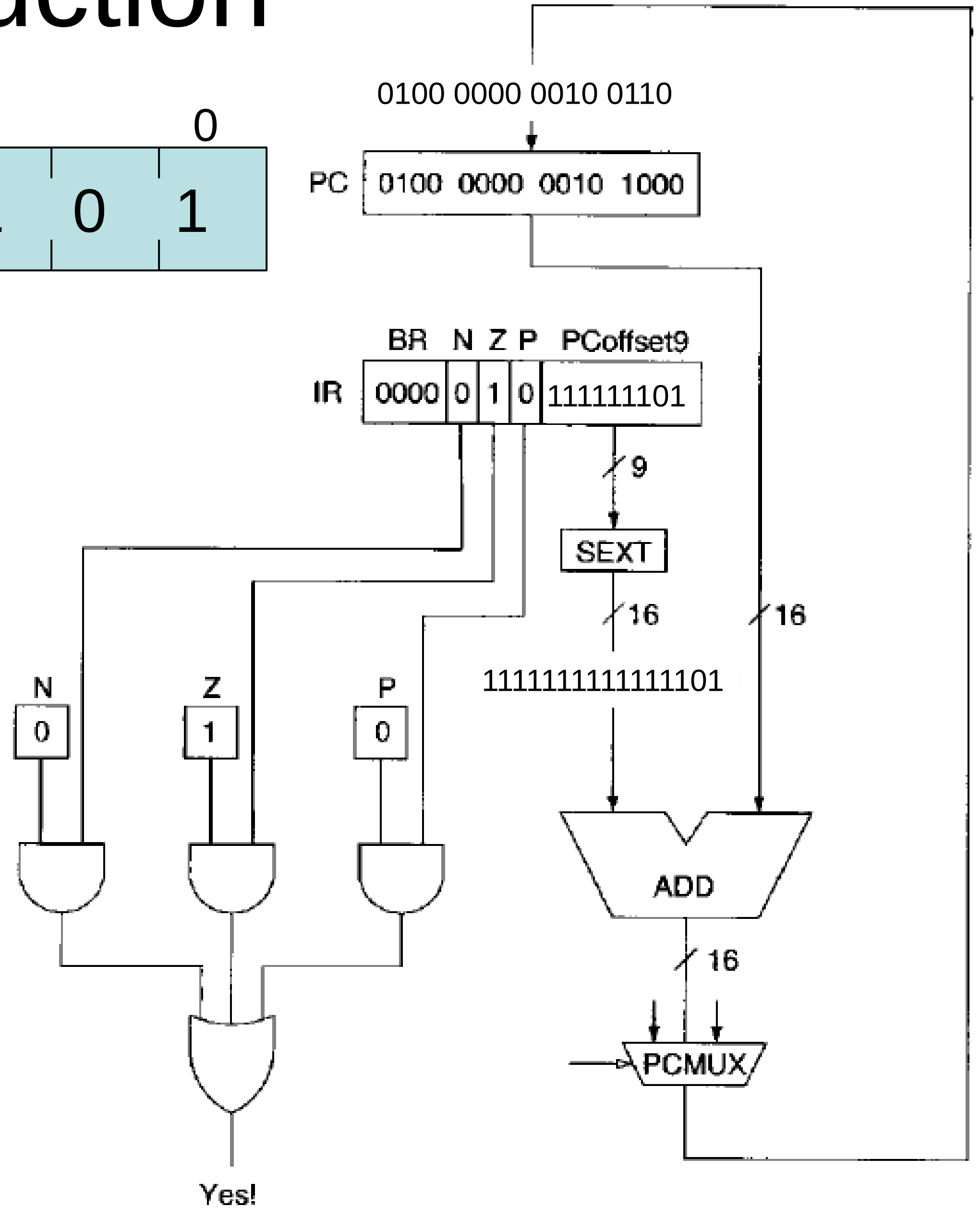
15			12	11		9	8								0
0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	1
opcode				Test Z bit			9-bit offset								

BRz instruction: if Z bit is set, then load the PC with the new address.

The new PC (for the branch taken case) is the incremented PC + the sign-extended 9-bit offset.

Here the PC is shown before incrementing: x4028.

The new PC (used only if the Z bit is set) is x4028 + 1 + xFFFD = x4026.



More on LC-3 BR

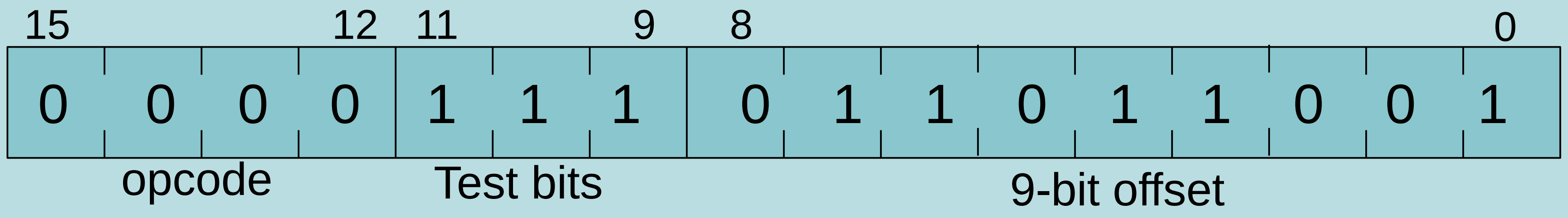
- What if the test bits in the instruction were set as follows?

15			12	11		9	8								0
0	0	0	0	1	1	1	0	1	1	0	1	1	0	0	1
opcode				Test bits			9-bit offset								

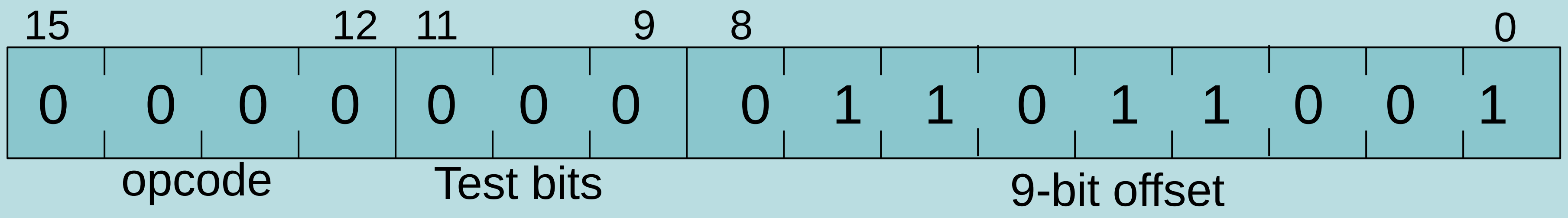
- What if the test bits in the instruction were set as follows?

15			12	11		9	8								0
0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1
opcode				Test bits			9-bit offset								

More on LC-3 BR



With all three bits set, the branch would be taken every time, so the instruction becomes an unconditional branch (goto).



With none of the bits set, the branch would never be taken, so the instruction becomes a no-op.

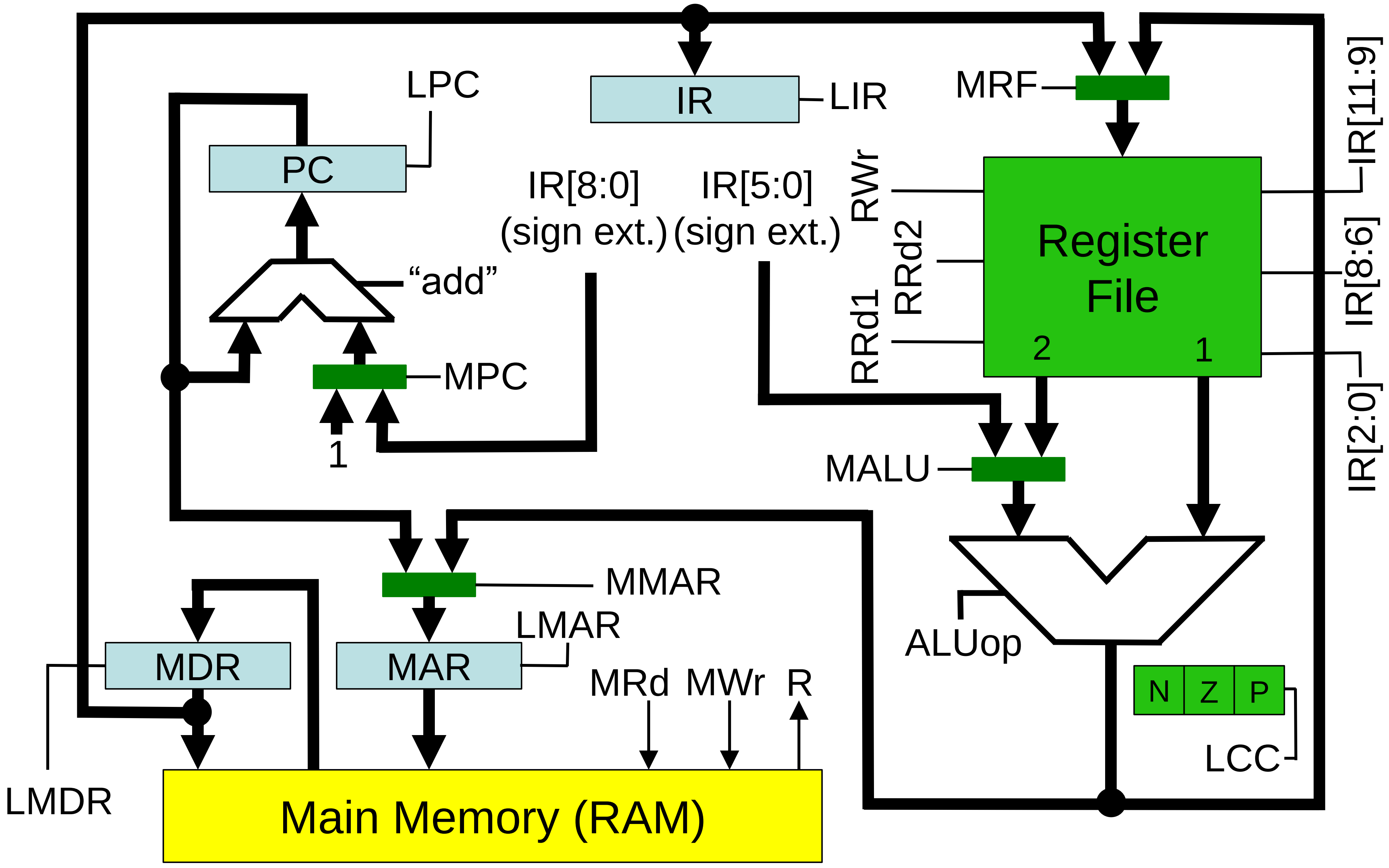
Outline

- More on LC-3 Instruction Set: Data Path Examples
  - Arithmetic/logical
  - Memory
  - Control
- Control Unit Design
  - Finite state machine

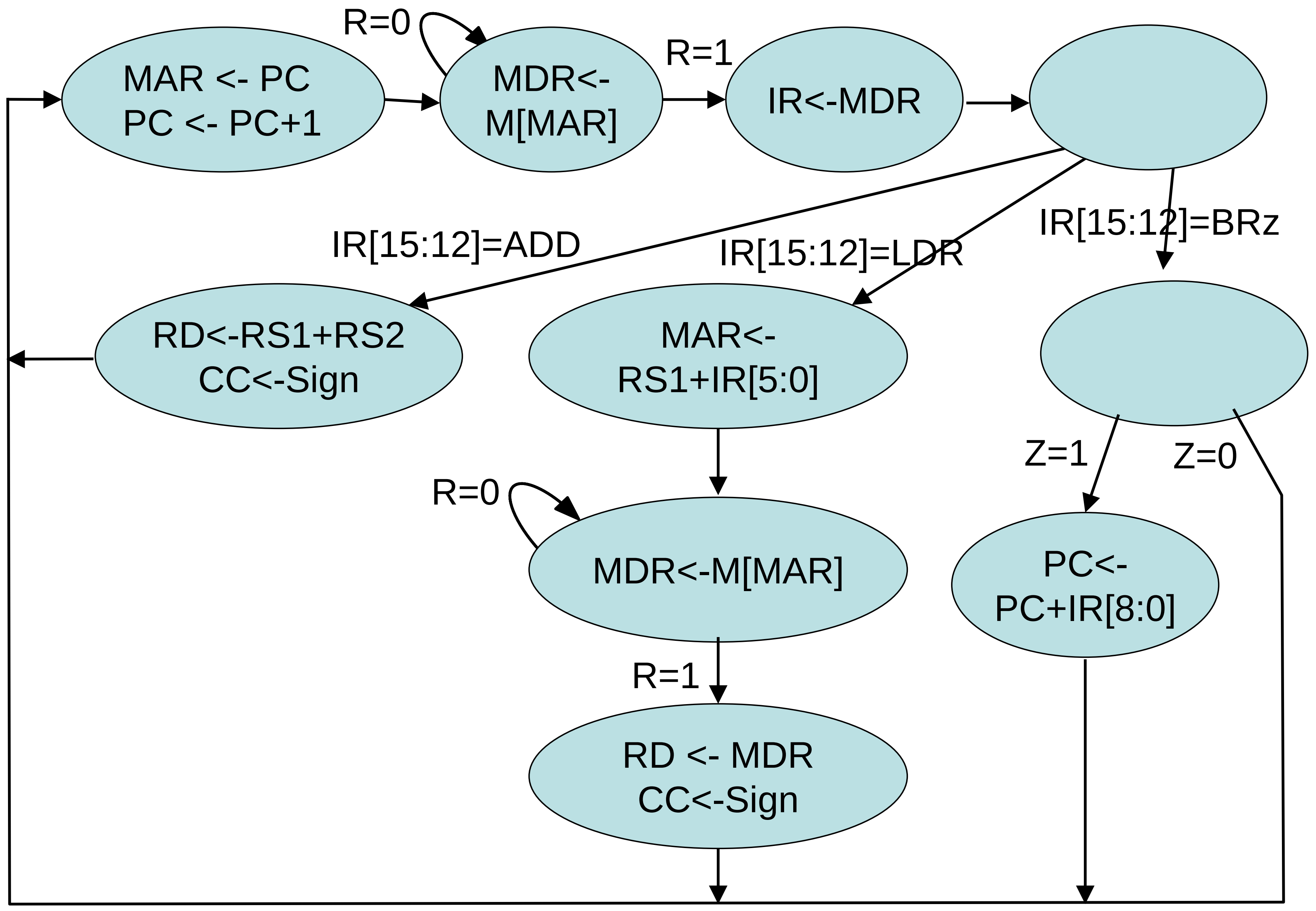
Partial LC-3 Data Path

(ADD, LDR, BRz instructions)

KEY: ⬇ 16 bits ↓ 1 bit 🟩 Multiplexer



LC-3 Control Unit (ADD, LDR, BRz Instructions)



Control and Status Signals

Control Signals (FSM outputs)

main memory

- MRd: read memory
- MWr: write memory

load register

- LPC: load PC
- LIR: load IR
- LMAR: load MAR
- LMDR: load MDR

mux control

- MPC: PC mux
- MMAR: MAR mux
- MRF: register file mux
- MALU: ALU mux

misc.

- ALUop: ALU operation
- LCC: load condition codes

Control Signals (cont.)

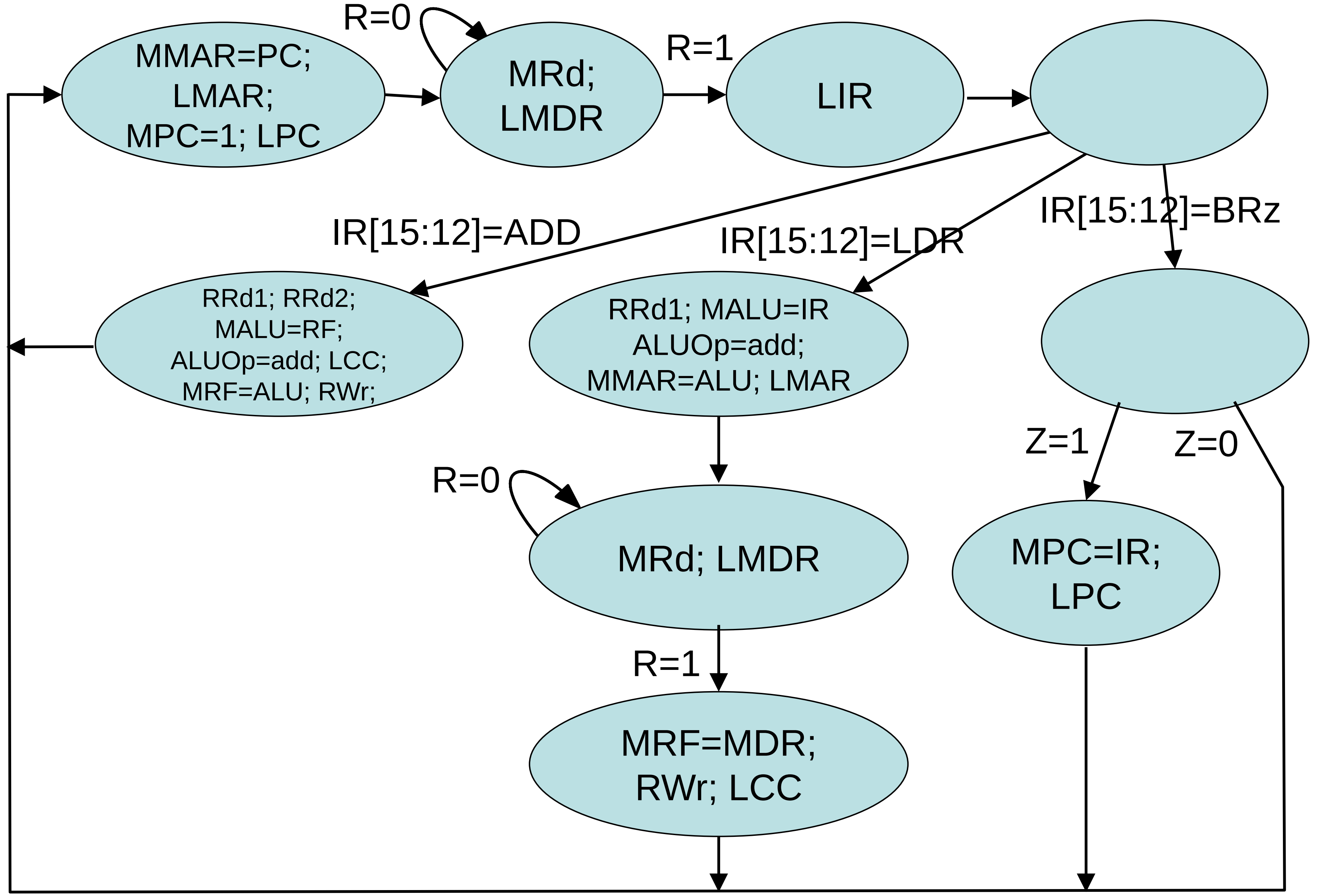
register file

- RRd1: read RF port 1
- RRd2: read RF port 2
- RWr: write RF

Status Signals (FSM inputs)

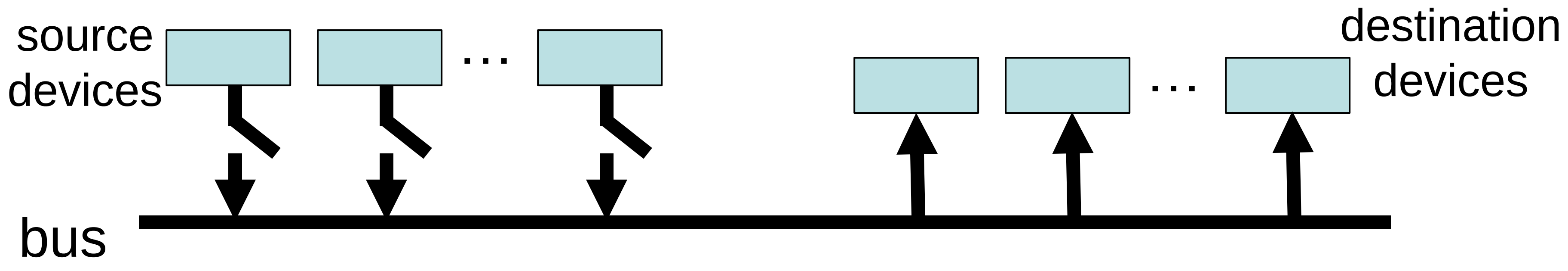
- R: Memory operation completed
- IR[15:12]: opcode
- N: N bit of condition code
- Z: Z bit of condition code
- P: P bit of condition code

LC-3 Control Unit (ADD, LDR, BRz Instructions)

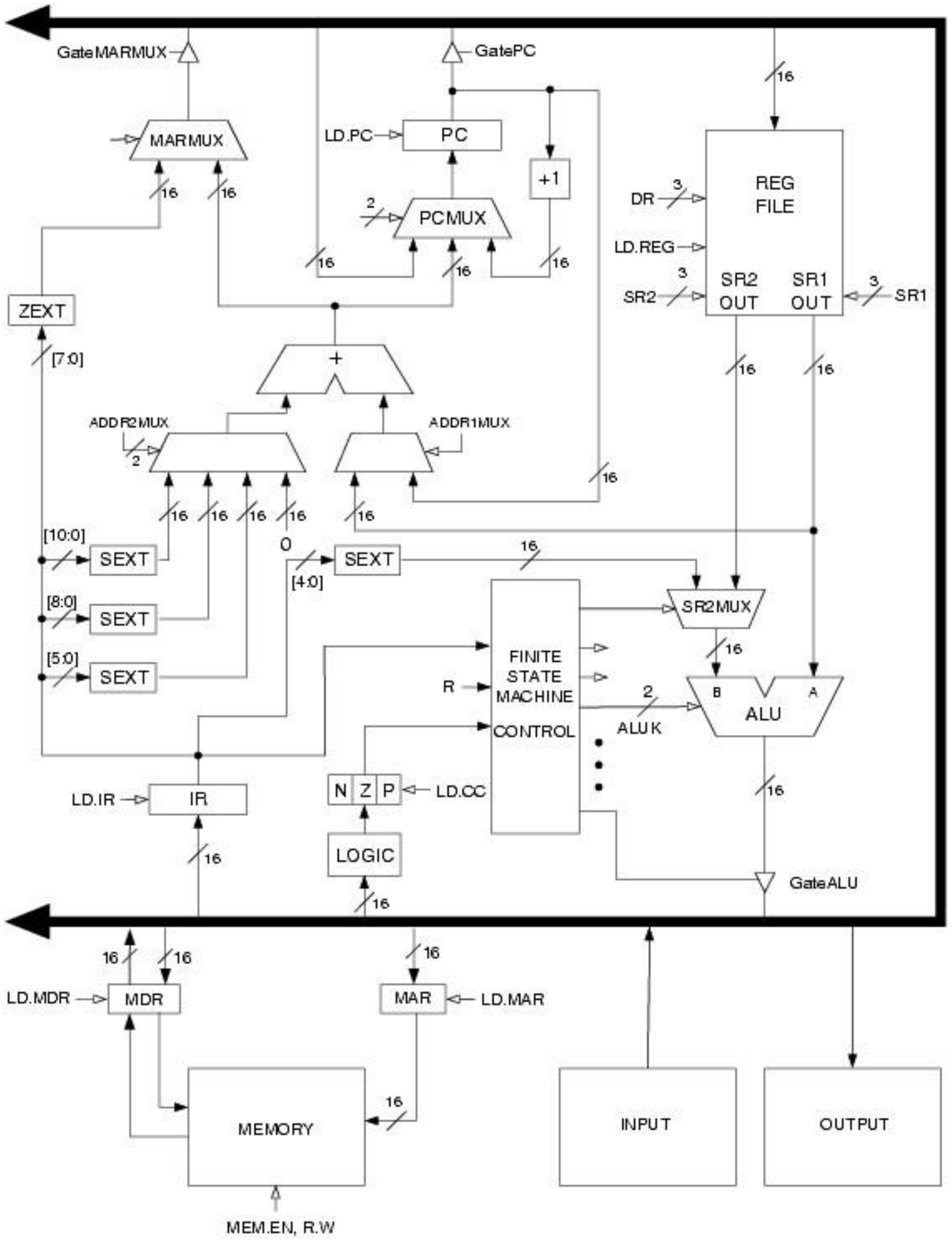

R=0
MMAR=PC; LMAR; MPC=1; LPC
MRd; LMDR
R=1
LIR
IR[15:12]=ADD
IR[15:12]=LDR
IR[15:12]=BRz
RRd1; RRd2; MALU=RF; ALUOp=add; LCC; MRF=ALU; RWr;
RRd1; MALU=IR ALUOp=add; MMAR=ALU; LMAR
Z=1
Z=0
R=0
MRd; LMDR
MPC=IR; LPC
R=1
MRF=MDR; RWr; LCC

Notes

- The data path and control unit shown so far only represent the realization of three machine instructions; the design process must be continued for the full instruction set
- It is convenient to define a “bus” in the data path to carry signals among multiple sources and destinations of data
  - Only one source device can put data on bus at one time!



LC-3 Data Path



Summary: CPU Design Process

- CPU = Data Path + Control Unit
- Data path design: determine what components you need
  - Memory visible to machine code programmer/compiler (PC, registers, condition codes)
  - Memory and memory interface (MAR, MDR)
  - Instruction Register (IR)
  - ALU(s)
- Control Unit: design finite state machine
  - Implement instruction processing cycle: Fetch instruction, decode instruction, evaluate address, fetch operands, execute operation, store result
  - Determines interconnection among the components in the data path design