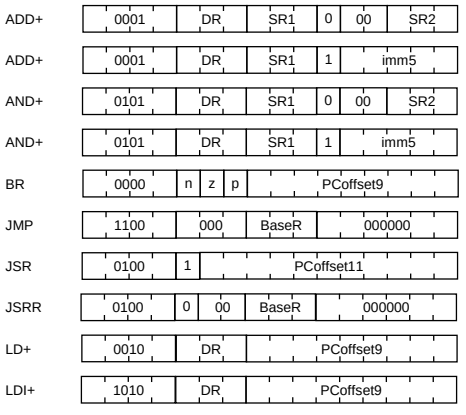
CS 135: Computer Architecture I

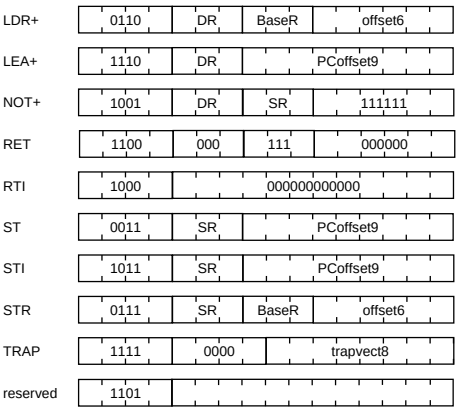
Instructor: Prof. Bhagi Narahari
Dept. of Computer Science
Course URL: www.seas.gwu.edu/~bhagiweb/cs135/

LC 3 Instruction Set

- **The Instruction set architecture (ISA) of the LC3**
  - **How is each instruction implemented by the control and data paths in the LC3**
  - **Programming in machine code**
  - **How are programs executed**
    - **Memory layout, programs in machine code**
- **Assembly programming**
  - **Assembly and compiler process**
  - **Assembly programming with simple programs**



+ Indicates instructions that modify condition codes



+ Indicates instructions that modify condition codes

LC-3 Overview: Memory and Registers

- **Memory**
  - address space: 2^{16} locations (16-bit addresses)
  - addressability: 16 bits
- **Registers**
  - temporary storage, accessed in a single machine cycle
    - accessing memory generally takes longer than a single cycle
  - eight general-purpose registers: R0 - R7
    - each 16 bits wide
    - how many bits to uniquely identify a register?
  - other registers
    - not directly addressable, but used by (and affected by) instructions
    - PC (program counter), condition codes

LC-3 Overview: Instruction Set

- **Opcodes**
  - 15 opcodes
  - *Operate* instructions: ADD, AND, NOT
  - *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
  - *Control* instructions: BR, JSR/JSRR, JMP, RTI, TRAP
  - some opcodes set/clear *condition codes*, based on result:
    - N = negative, Z = zero, P = positive (> 0)
- **Data Types**
  - 16-bit 2's complement integer
- **Addressing Modes**
  - How is the location of an operand specified?
  - non-memory addresses: *immediate*, *register*
  - memory addresses: *PC-relative*, *indirect*, *base+offset*

Operate Instructions

- **Only three operations: ADD, AND, NOT**
- **Source and destination operands are registers**
  - These instructions ***do not*** reference memory.
  - ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.
- **Will show dataflow diagram with each instruction.**
  - illustrates ***when*** and ***where*** data moves to accomplish the desired operation

Data Movement Instructions

- GPR ↔ Memory
- GPR ↔ I/O Devices
- GPR ← Memory ???
- Memory ← GPR ???

Addressing Modes

- Where can operands be found?

1

2

3

Data Movement Instructions

- Load -- read data from memory to register
  - LD: PC-relative mode
  - LDR: base+offset mode
  - LDI: indirect mode
- Store -- write data from register to memory
  - ST: PC-relative mode
  - STR: base+offset mode
  - STI: indirect mode
- Load effective address -- compute address, save in register
  - LEA: immediate mode
  - *does not access memory*

PC-Relative Addressing Mode

- Want to specify address directly in the instruction
  - But an address is 16 bits, and so is an instruction!
  - After subtracting 4 bits for opcode and 3 bits for register, we have 9 bits available for address.
- Solution:
  - Use the 9 bits as a *signed offset* from the current PC.

- 9 bits: $-256 \le \text{offset} \le +255$
- Can form any address X, such that: $PC - 256 \le X \le PC + 255$

- Remember that PC is incremented as part of the FETCH phase;
- This is done before the EVALUATE ADDRESS stage.

Control Instructions

- Used to alter the sequence of instructions (by changing the Program Counter)
- Conditional Branch
  - branch is *taken* if a specified condition is true
    - signed offset is added to PC to yield new PC
  - else, the branch is *not taken*
    - PC is not changed, points to the next sequential instruction
- Unconditional Branch (or Jump)
  - always changes the PC
- TRAP
  - changes PC to the address of an OS "service routine"
  - routine will return control to the next instruction (after TRAP)

Condition Codes

- LC-3 has three condition code registers:
  - N -- negative
  - Z -- zero
  - P -- positive (greater than zero)
- Set by any instruction that writes a value to a register (ADD, AND, NOT, LD, LDR, LDI, LEA)
- Exactly one will be set at all times
  - Based on the last instruction that altered a register

Branch Instruction

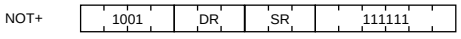
- Branch specifies one or more condition codes.
- If the set bit is specified, the branch is taken.
  - PC-relative addressing: target address is made by adding signed offset (IR[8:0]) to current PC.
  - Note: PC has already been incremented by FETCH stage.
  - Note: Target must be within 256 words of BR instruction.
- If the branch is not taken, the next sequential instruction is executed.

Operate Instructions

- Only three operations: ADD, AND, NOT
- Source and destination operands are registers
  - These instructions *do not* reference memory.
  - ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.
- Will show dataflow diagram with each instruction.
  - illustrates *when* and *where* data moves to accomplish the desired operation

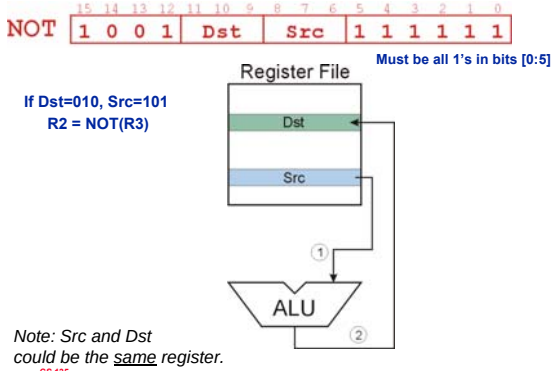
Operate Instructions

- NOT



- Addressing mode?

NOT (Register)

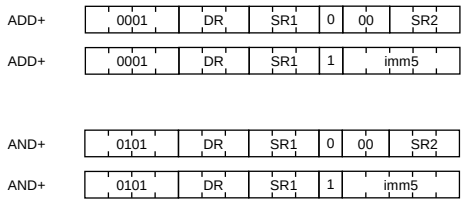


Note: Src and Dst could be the same register.

CS 135

Operate Instructions

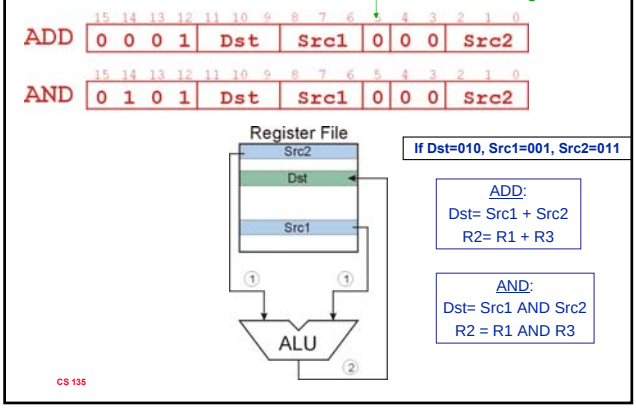
- **ADD, AND**



- **Addressing Mode?**

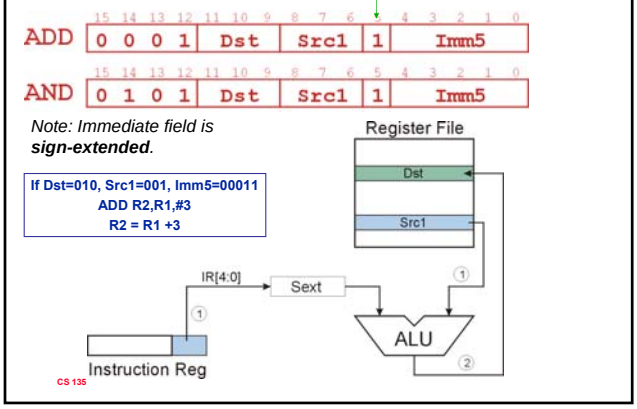
CS 135

ADD/AND (Register)



CS 135

ADD/AND (Immediate)



CS 135

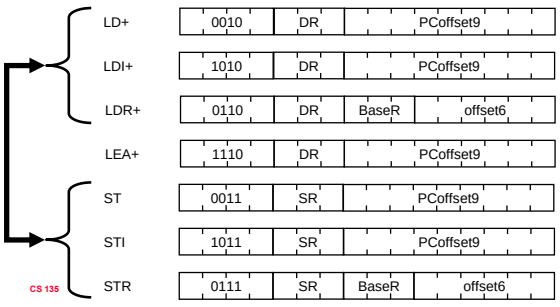
Using Operate Instructions

- With only ADD, AND, NOT…
  - How do we subtract?
  - How do we OR?
  - How do we copy from one register to another?
  - How do we initialize a register to zero?

Data Movement Instructions

- GPR ↔ Memory
- GPR ↔ I/O Devices
- GPR ← Memory ???
- Memory ← GPR ???

Data Movement Instructions



Addressing Modes

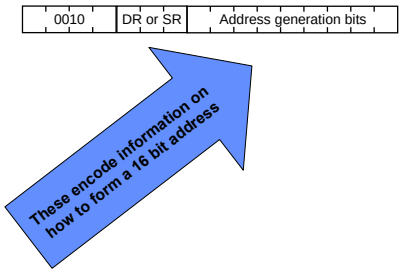
- Where can operands be found?

1

2

3

Basic Format



Data Movement Instructions

- **Load -- read data from memory to register**
  - **LD: PC-relative mode**
  - **LDR: base+offset mode**
  - **LDI: indirect mode**
- **Store -- write data from register to memory**
  - **ST: PC-relative mode**
  - **STR: base+offset mode**
  - **STI: indirect mode**
- **Load effective address -- compute address, save in register**
  - **LEA: immediate mode**
  - ***does not access memory***

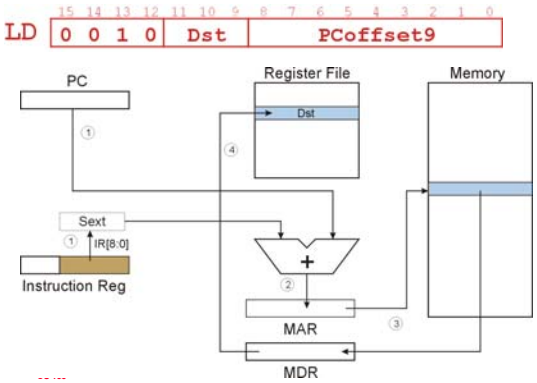
PC-Relative Addressing Mode

- **Want to specify address directly in the instruction**
  - **But an address is 16 bits, and so is an instruction!**
  - **After subtracting 4 bits for opcode and 3 bits for register, we have 9 bits available for address.**
- **Solution:**
  - **Use the 9 bits as a *signed offset* from the current PC.**

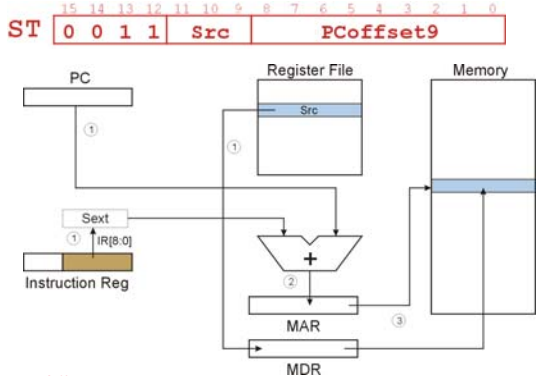
- **9 bits:** $-256 \le \text{offset} \le +255$
- **Can form any address X, such that:** $PC - 256 \le X \le PC + 255$

- **Remember that PC is incremented as part of the FETCH phase;**
- **This is done before the EVALUATE ADDRESS stage.**

LD (PC-Relative)



ST (PC-Relative)

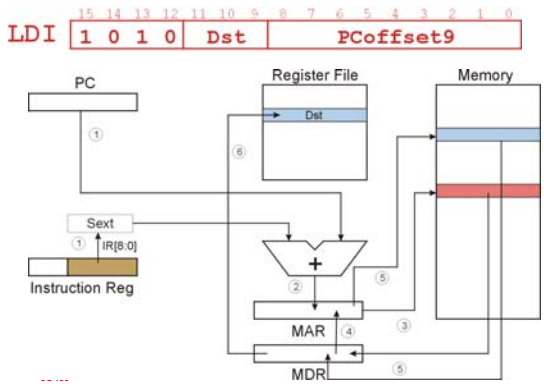


Indirect Addressing Mode

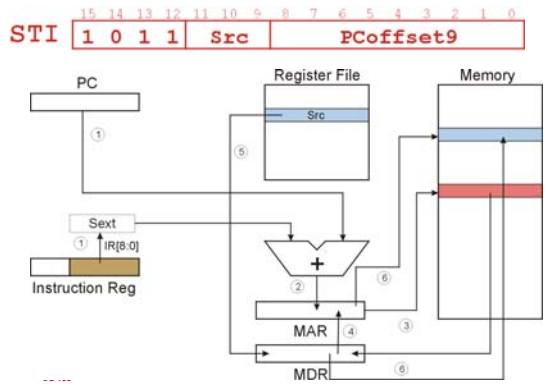
- With PC-relative mode, can only address data within 256 words of the instruction.
  - What about the rest of memory?
- Solution #1:
  - Read address from memory location, then load/store to that address.
- First address is generated from PC and IR (just like PC-relative addressing), then content of that address is used as target for load/store.

CS 135

LDI (Indirect)



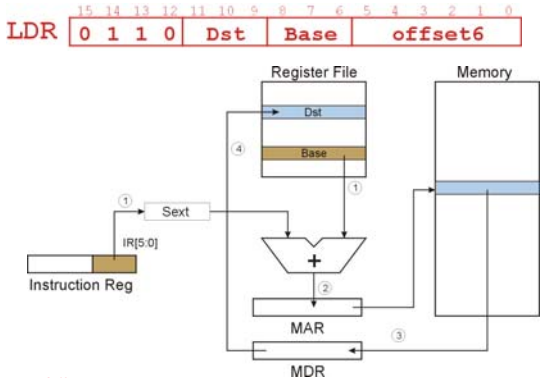
STI (Indirect)



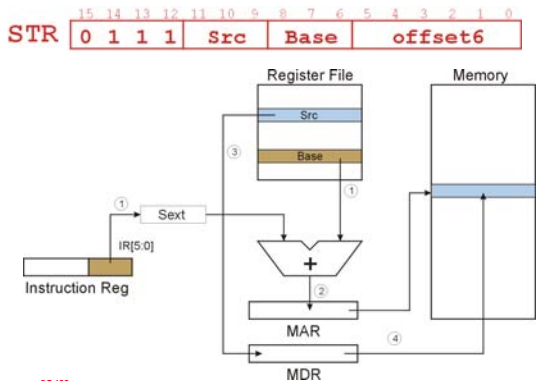
Base + Offset Addressing Mode

- **With PC-relative mode, can only address data within 256 words of the instruction.**
  - **What about the rest of memory?**
- **Solution #2:**
  - **Use a register to generate a full 16-bit address.**
- **4 bits for opcode, 3 for src/dest register, 3 bits for *base* register -- remaining 6 bits are used as a *signed offset*.**
  - Offset is *sign-extended* before adding to base register.

LDR (Base+Offset)



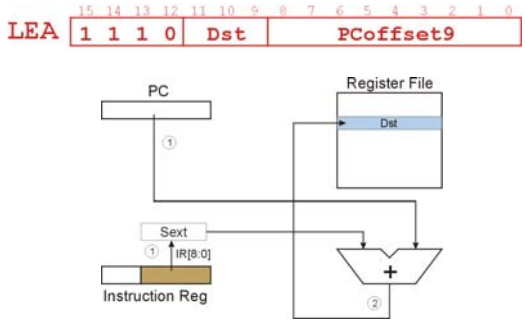
STR (Base+Offset)



Load Effective Address

- **Computes address like PC-relative (PC plus signed offset) and stores the result into a register.**
- **Note: The *address* is stored in the register, not the contents of the memory location.**

LEA (Immediate)



CS 135

Example

Address	Instruction	Comments
x30F6	1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1	*R1 ← PC – 3 = x30F4*
x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	*R2 ← R1 + 14 = x3102*
x30F8	0 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1	*M[PC - 5] ← R2* *M[x30F4] ← x3102*
x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0*
x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	*R2 ← R2 + 5 = 5*
x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	*M[R1+14] ← R2* *M[x3102] ← 5*
x30FC	1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1	*R3 ← M[M[x30F4]]* *R3 ← M[x3102]* *R3 ← 5*

opcode

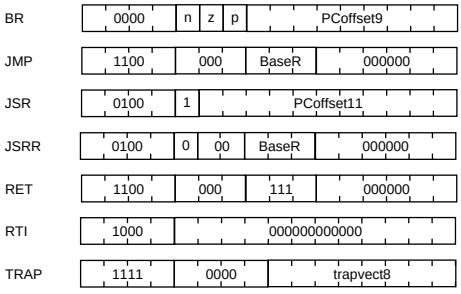
CS 135

Control Instructions

- **Used to alter the sequence of instructions (by changing the Program Counter)**
- **Conditional Branch**
  - **branch is *taken* if a specified condition is true**
    - **signed offset is added to PC to yield new PC**
  - **else, the branch is *not taken***
    - **PC is not changed, points to the next sequential instruction**
- **Unconditional Branch (or Jump)**
  - **always changes the PC**
- **TRAP**
  - **changes PC to the address of an OS "service routine"**
  - **routine will return control to the next instruction (after TRAP)**

CS 135

Control Instructions



CS 135

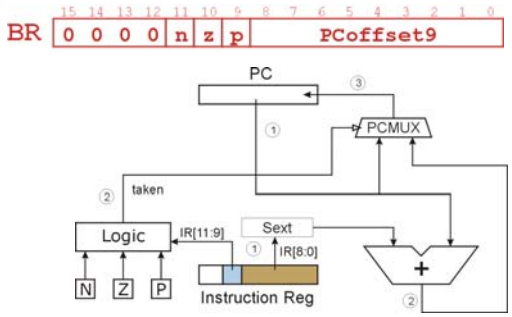
Condition Codes

- LC-3 has three condition code registers:
  - N -- negative
  - Z -- zero
  - P -- positive (greater than zero)
- Set by any instruction that writes a value to a register (ADD, AND, NOT, LD, LDR, LDI, LEA)
- Exactly one will be set at all times
  - Based on the last instruction that altered a register

Branch Instruction

- Branch specifies one or more condition codes.
- If the set bit is specified, the branch is taken.
  - PC-relative addressing: target address is made by adding signed offset (IR[8:0]) to current PC.
  - Note: PC has already been incremented by FETCH stage.
  - Note: Target must be within 256 words of BR instruction.
- If the branch is not taken, the next sequential instruction is executed.

BR (PC-Relative)



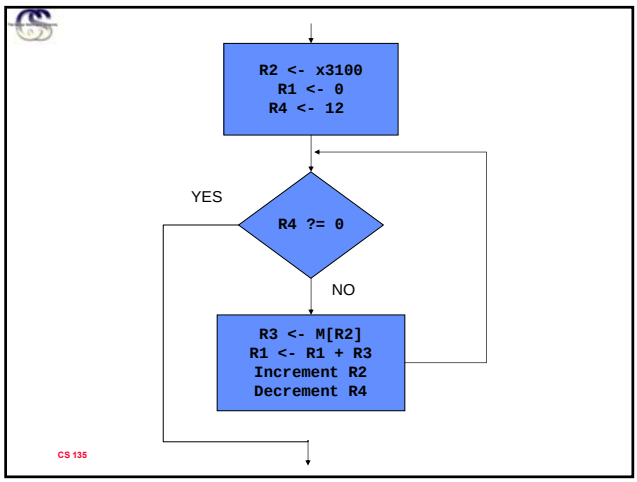
What happens if bits [11:9] are all zero? All one?

Using Branch Instructions

- Compute sum of 12 integers.

Numbers start at location x3100. Program starts at location x3000.

- Add numbers from location x3100 to x311B
- Store first address in R2
- R4 has "counter" – counts down from 12 to 0
- R1 will store the running Sum



CS 135

Program

```
x3000 R2 <- x3100
x3001 R4 <- 0
x3002 R1 <- 0
x3003 R4 <- 12
x3004 BRz x300A
x3005 R3 <- M[R2]
x3006 R1 <- R1 + R3
x3007 R2 <- R2 + 1
x3008 R4 <- R4 - 1
x3009 BRnzp x3004
```

CS 135

Program

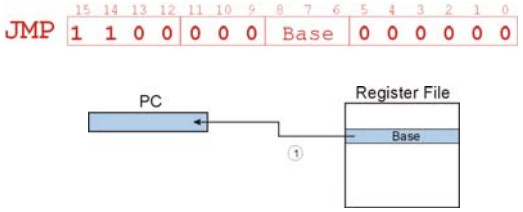
		opcode
x3000 R2 <- x3100	LEA	1110010011111111
x3001 R1 <- 0	AND	0101001011100000
x3002 R4 <- 0	AND	0101100010100000
x3003 R4 <- 12	ADD	0001100010101100
x3004 BRz x300A	BRz	0000010000000101
x3005 R3 <- M[R2]	LDR	0110011010000000
x3006 R1 <- R1 + R3	ADD	0001001001000011
x3007 R2 <- R2 + 1	ADD	0001010010100001
x3008 R4 <- R4 - 1	ADD	0001100100111111
x3009 BRnzp x3004	BRnzp	0000111111111010

dest, Immediate, source

CS 135

JMP (Register)

- Jump is an unconditional branch -- ***always*** taken.
  - Target address is the contents of a register.
  - Allows any target address.



CS 135

TRAP Instruction

- Modern computers contain hardware and software protection schemes to prevent user programs from accidentally (or maliciously) interfering with proper system function.
- Suffice it to say, we need a way to communicate with the operating system

TRAP

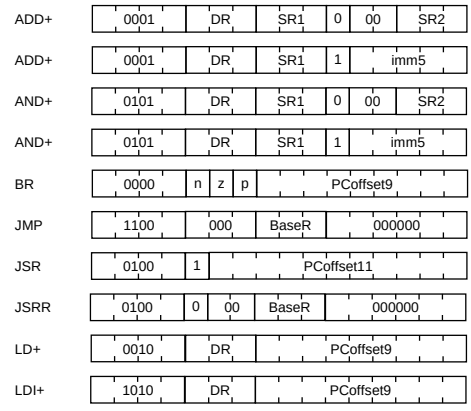
- Calls a service routine, identified by 8-bit "trap vector."

vector	*routine*
x23	input a character from the keyboard
x21	output a character to the monitor
x25	halt the program

- When routine is done, PC is set to the instruction following TRAP.
- (We'll talk about how this works later.)

The LC-3 ISA: summary

- 16 bit instructions and data
- 2's complement data type
- Operate/ALU instructions: ADD, NOT, AND
- Data movement Inst: Load and Store
  - Addressing mode: PC-relative, Indirect, Register/Base+Offset,Immediate
- Transfer of control instructions
  - Branch – using condition code registers
  - Jump – unconditional branch
  - Traps, Subroutine calls – discuss later
- Let's take a peek at the LC3 datapath and controller design



+ Indicates instructions that modify condition codes

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			

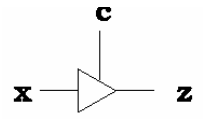
+ Indicates instructions that modify condition codes

Von Neumann Model: Outline

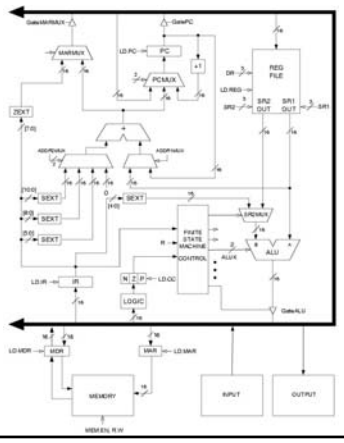
- **Basic Components**
  - **Memory, Processing Unit, Input & Output, Control Unit**
- **LC-3: An Example von Neumann Machine**
- **Instruction Processing**
  - **The Instruction, The Instruction Cycle**
    - **Fetch, Decode, Evaluate Address, Fetch Operands, Execute, Store Result**
  - **Changing the Sequence of Execution**
    - **Branches and Jumps**
  - **Stopping the Computer**

Another device: tri-state buffer

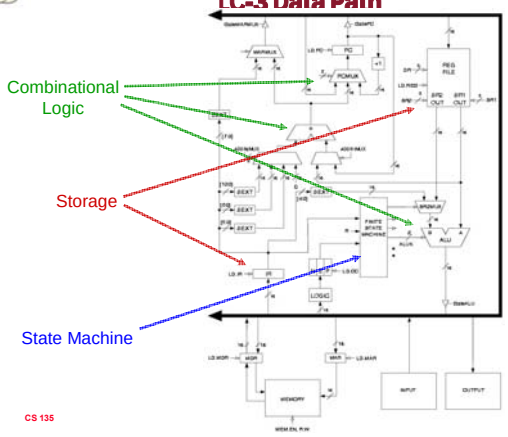
- **inputs to the bus are "tri-state devices," that only place a signal on the bus when they are enabled**
- **Tri-state buffer controls when current passes through the line**
  - **A true "open switch"**
  - **When control signal c=1 then x=z else "open switch"**

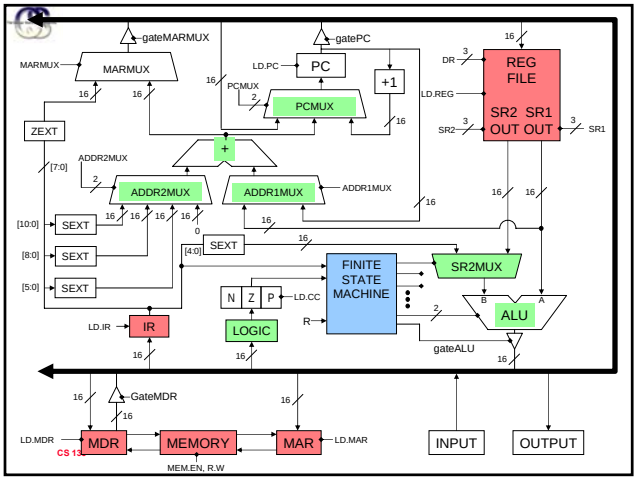


Filled arrow
= info to be processed.
Unfilled arrow
= control signal.



LC-3 Data Path





Data Path Components

- **Global bus**
  - special set of wires that carry a 16-bit signal to many components
  - inputs to the bus are "tri-state devices," that only place a signal on the bus when they are enabled
  - only one (16-bit) signal should be enabled at any time
    - control unit decides which signal "drives" the bus
  - any number of components can read the bus
    - register only captures bus data if it is write-enabled by the control unit
- **Memory**
  - Control and data registers for memory and I/O devices
  - memory: MAR, MDR (also control signal for read/write)

Data Path Components

- **ALU**
  - Accepts inputs from register file and from sign-extended bits from IR (immediate field).
    - Bit 5 of LC3 instruction determines this
  - Output goes to bus.
    - used by condition code logic, register file, memory
  - Function to apply: determined by opcode – need 2 bits ALUK
- **Register File**
  - Two read addresses (SR1, SR2), one write address (DR)
  - Input from bus
    - result of ALU operation or memory read
  - Two 16-bit outputs
    - used by ALU, PC, memory address
    - data for store instructions passes through ALU

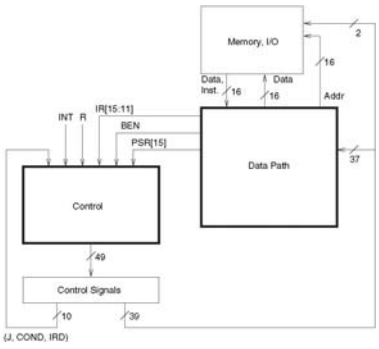
Data Path Components

- **PC and PCMUX**
  - Three inputs to PC, controlled by PCMUX
    1. PC+1 – FETCH stage
    2. Address adder – BR, JMP
    3. bus – TRAP (discussed later)

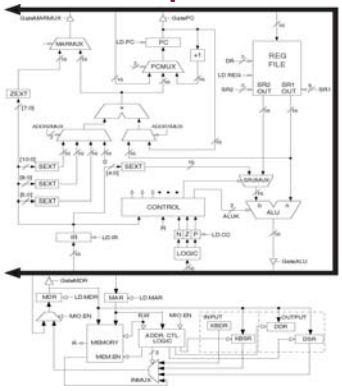
- **MAR and MARMUX**
  - Two inputs to MAR, controlled by MARMUX
    1. Address adder – LD/ST, LDR/STR
    2. Zero-extended IR[7:0] -- TRAP (discussed later)

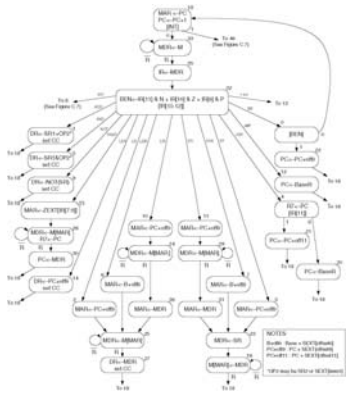
Data Path Components

- **Condition Code Logic**
  - Looks at value on bus and generates N, Z, P signals
  - Registers set only when control unit enables them (LD.CC)
    - only certain instructions set the codes (ADD, AND, NOT, LD, LDI, LDR, LEA)
- **Control Unit – Finite State Machine**
  - On each machine cycle, changes control signals for next phase of instruction processing
    - **who drives the bus?** (GatePC, GateALU, …)
    - **which registers are write enabled?** (LD.IR, LD.REG, …)
    - **which operation should ALU perform?** (ALUK)
    - …
  - Logic includes decoder for opcode, etc.



LC3 data path





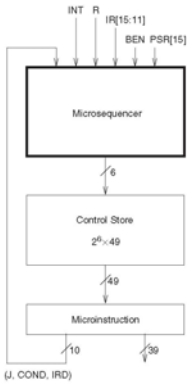
CS 135

Implementing the Control Logic

- Given the state diagram one can implement the controller in many ways
  - 52 states
  - Each needs 39 control lines plus 10 to determine next state = 49 control lines
- What should controller do ?
  - Generate the 49 control signals at each cycle
- Implement this as a Microprogram
  - Use 6 bit address to get the microinstruction
  - Start state and progress through states based on microinstruction

CS 135

Microprogrammed Implementation



CS 135

Course Summary

- Datatypes of machines: Number Representation
  - 2's complement integers, Floating point
  - Arithmetic on 2's complement
  - Logic operations
- Digital logic: devices to build the circuits
  - CMOS transistor is the starting point
  - Basic logic gates: AND, OR, NOT, NAND, etc.
  - Combinational logic 'blocks': MUX, Decoder, PLA
  - Sequential Logic: storage element, finite state machines
  - Putting it all together to build a simple processor- LC3
- Von Neumann Model of computing
- Instruction set architecture (ISA) of LC3
  - Instructions of a processor – how program execution takes place
  - Addressing modes to data movement, branches, operations
  - Programming in machine language

CS 135