

Chapter 5

The LC-3

Instruction Set Architecture

ISA = All of the *programmer-visible* components and operations of the computer

- **memory organization**
 - address space -- how many locations can be addressed?
 - addressability -- how many bits per location?
- **register set**
 - how many? what size? how are they used?
- **instruction set**
 - opcodes
 - data types
 - addressing modes

ISA provides all information needed for someone that wants to write a program in **machine language** (or translate from a high-level language to machine language).

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

Dataflow diagrams

Will show **dataflow diagram** with each instruction.

- illustrates when and where data moves to accomplish the desired operation

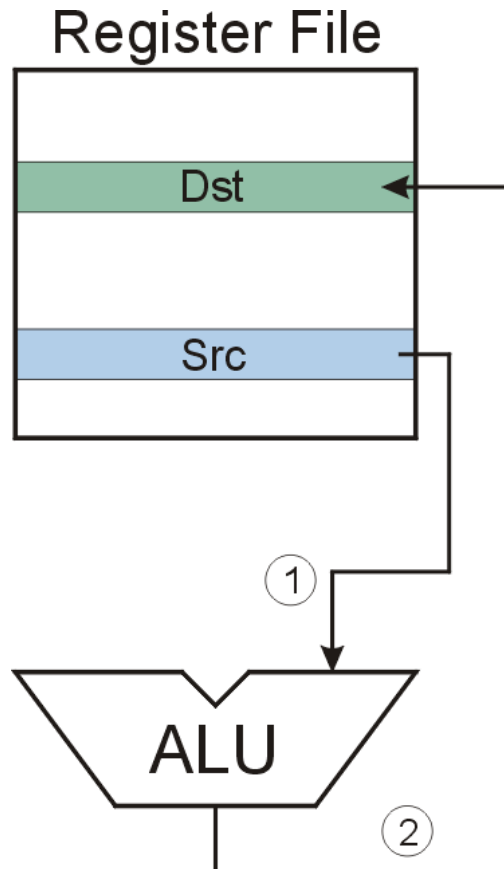
Components in data flow diagrams:

- **Registers:** each register can hold 16 bits in LC-3. Several
- **Register File:** contains eight 16-bit registers
- **ALU:** combinational (no storage). Two inputs, one output, functionality can be selected.
- **SEXT:** combinational. Sign extension from 5 to 16 bits.
- **Memory:** 2¹⁶ words. Addressed by 16 bit MAR and MDR holds data

How components are implemented?

- Last third of the class

NOT (Register)

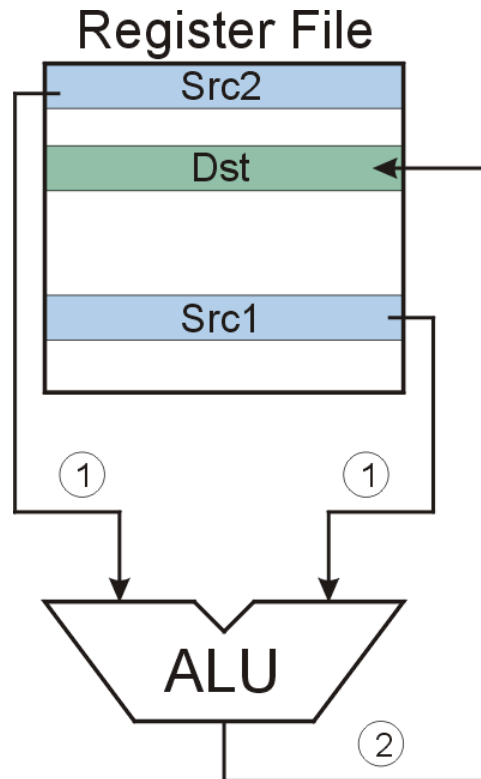


Assembly Ex:
NOT R3, R2

Note: Src and Dst could be the same register.

ADD/AND (Register)

this zero means "register mode"



Assembly Ex:
Add R3, R1, R3

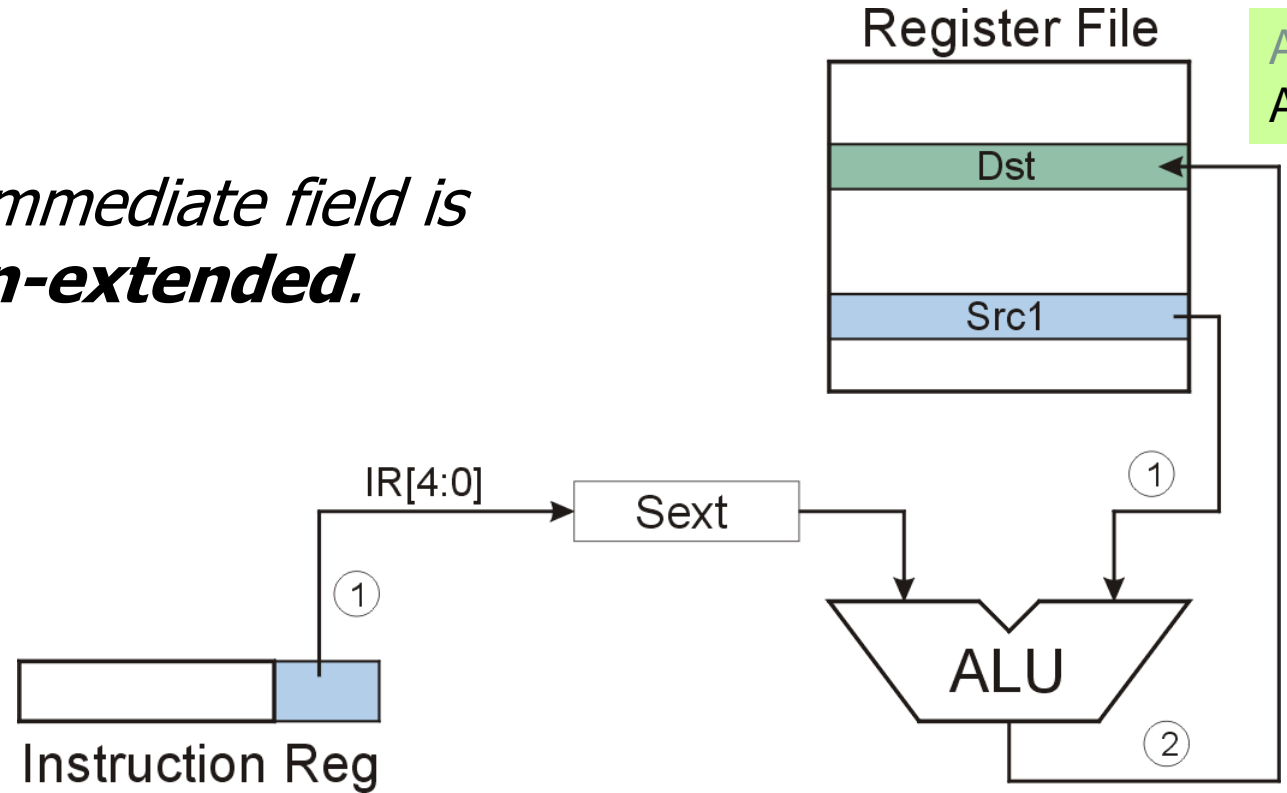
ADD/AND (Immediate)

this one means "immediate mode"



Assembly Ex:
Add R3, R3, #1

Note: Immediate field is **sign-extended**.



Using Operate Instructions

With only ADD, AND, NOT...

- How do we subtract? Hint: Negate and add
- How do we OR? Hint: Demorgan's law
- How do we copy from one register to another?
- How do we initialize a register to zero?

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.

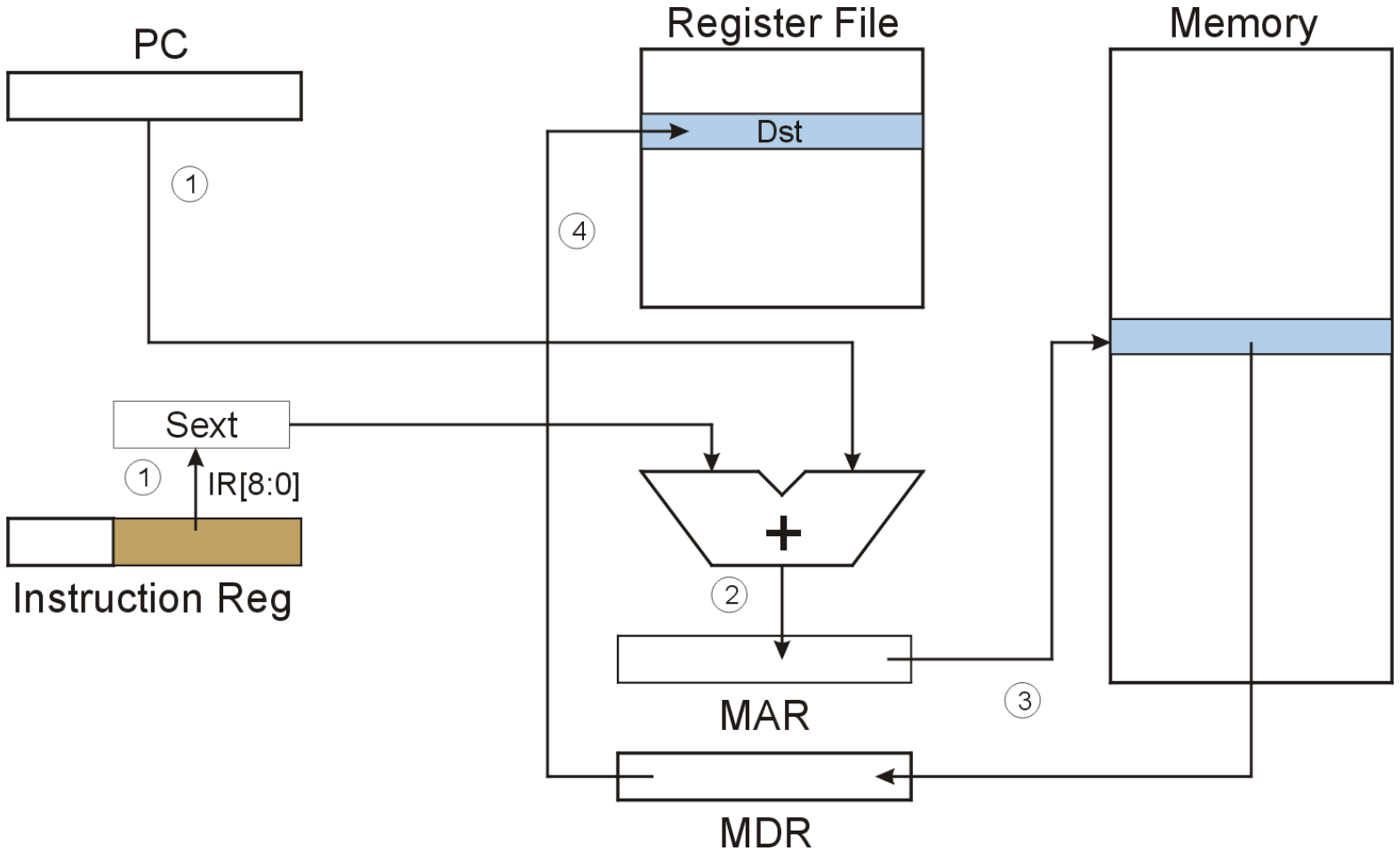
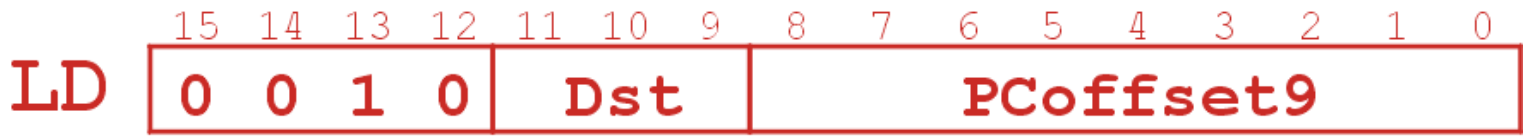
9 bits: $-256 \leq \text{offset} \leq +255$

Can form any address X, such that: $PC - 256 \leq X \leq PC + 255$

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

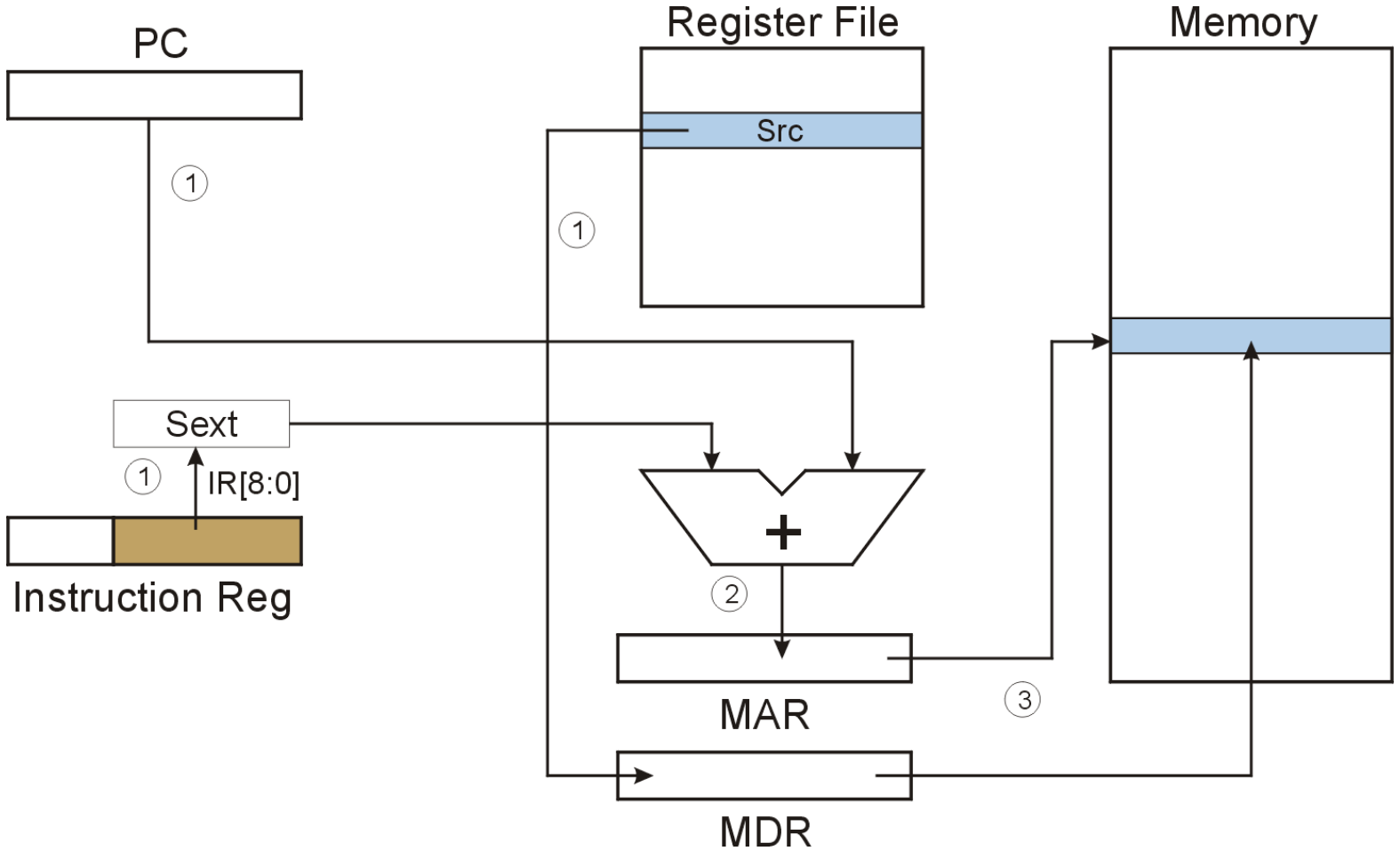
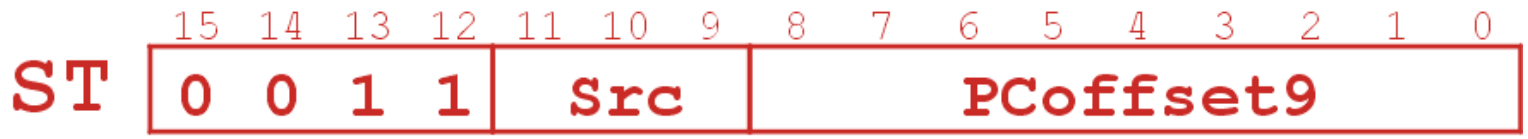
Assembly Ex:
LD R1, Label1

LD (PC-Relative)



Assembly Ex:
ST R1, Label2

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?

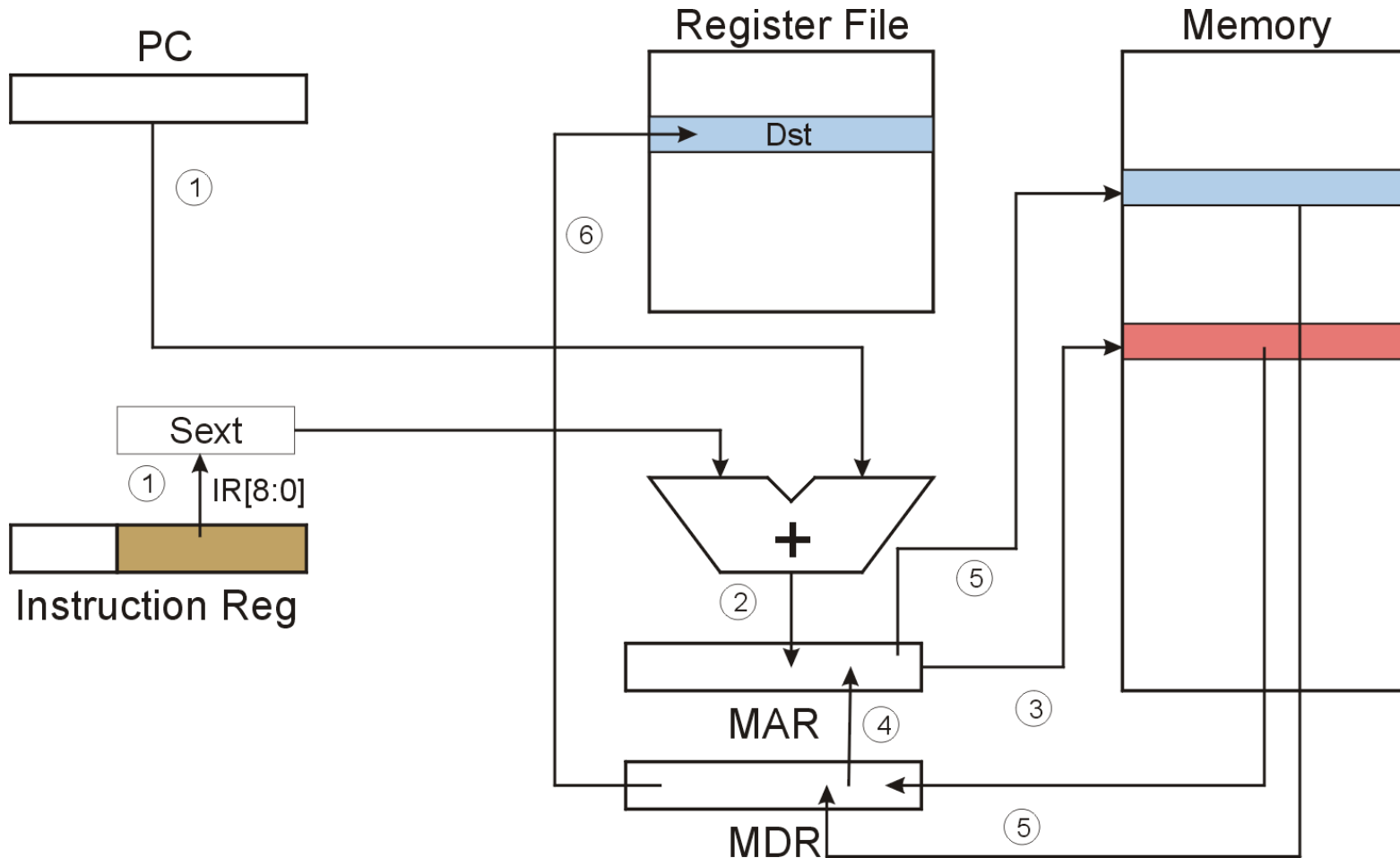
Solution #1:

- 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.

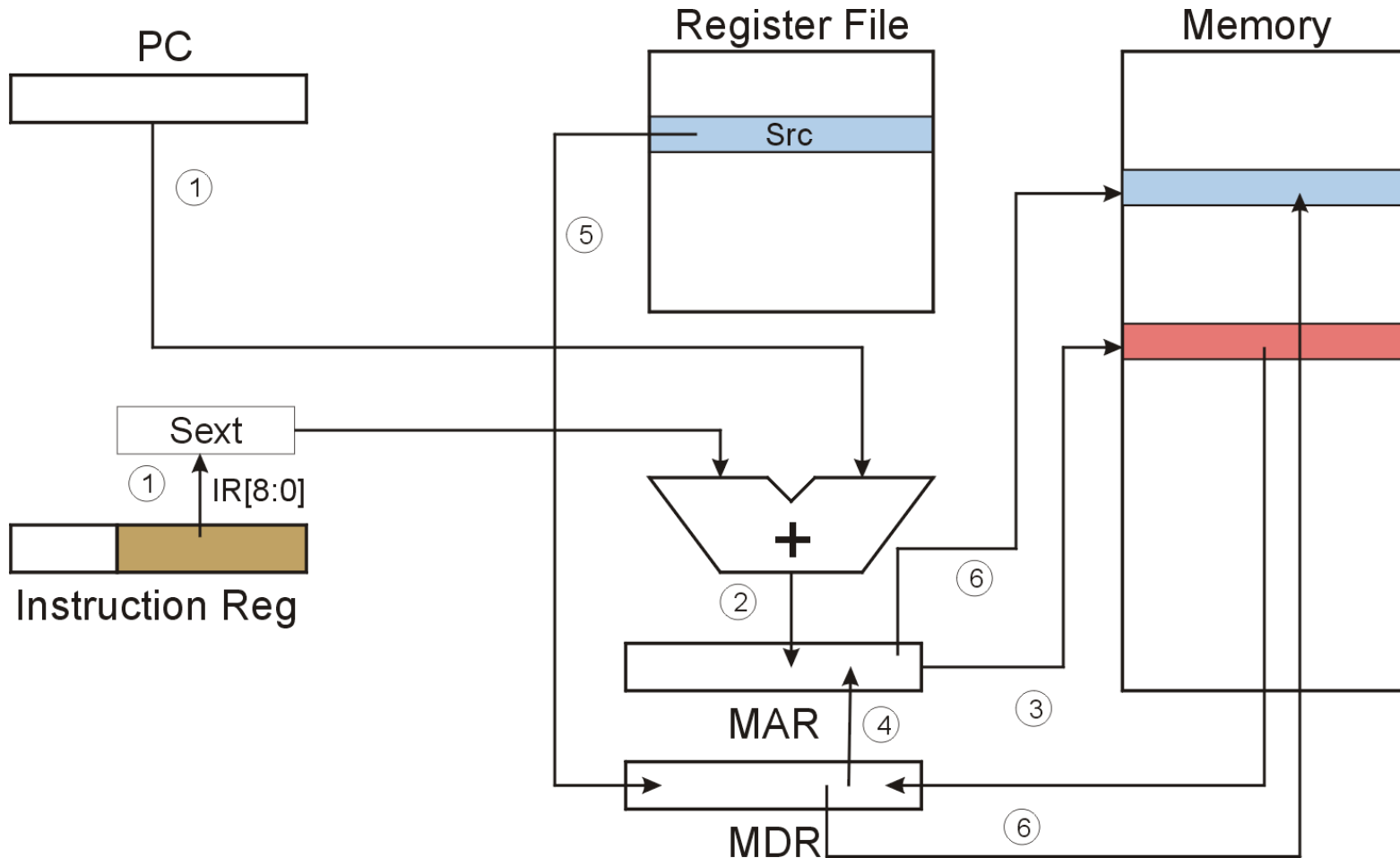
LDI (Indirect)

Assembly Ex:
LDI R4, Adr



STI (Indirect)

Assembly Ex:
STI R4, Adr



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:

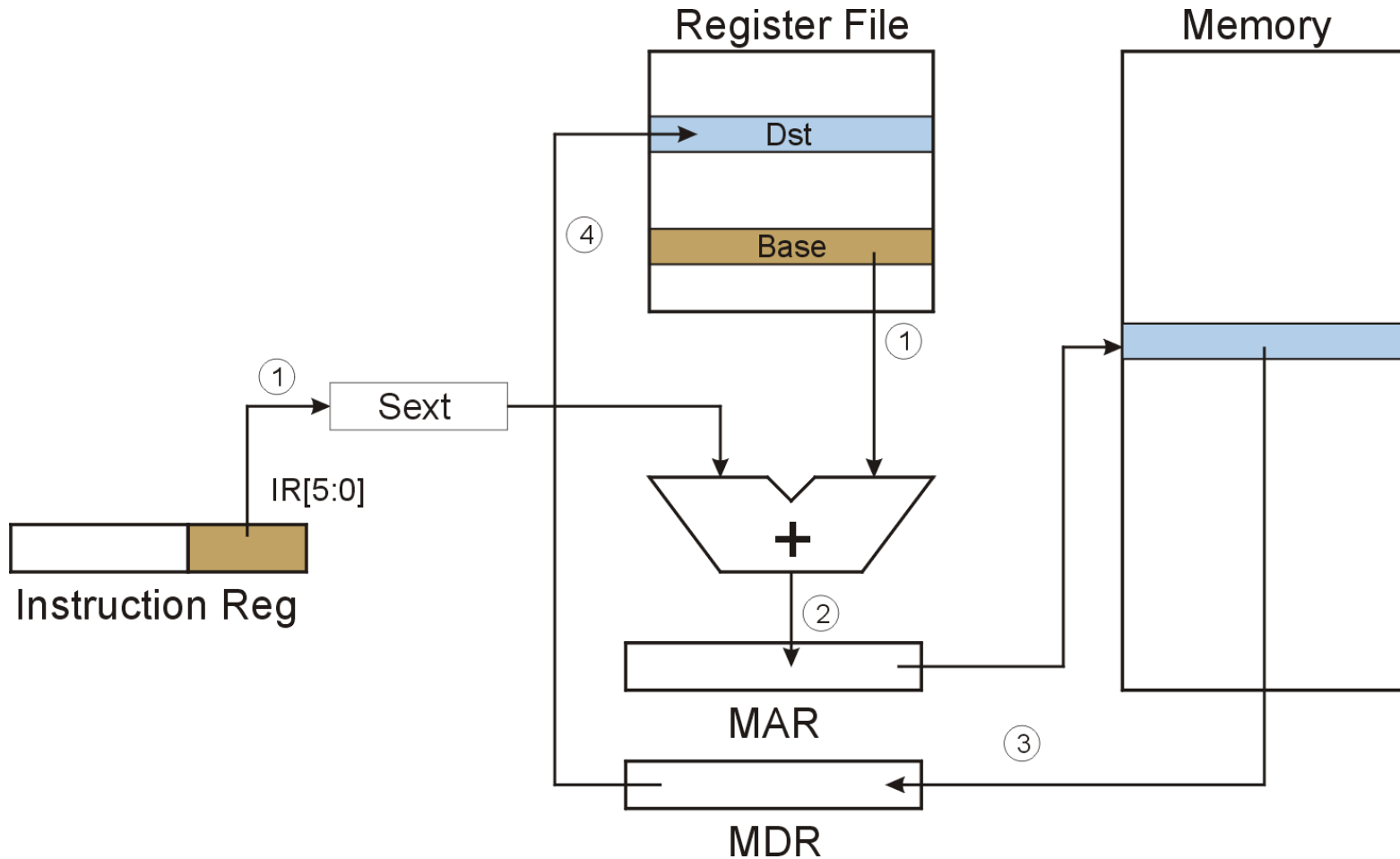
- Use a register to generate a full 16-bit address.

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.

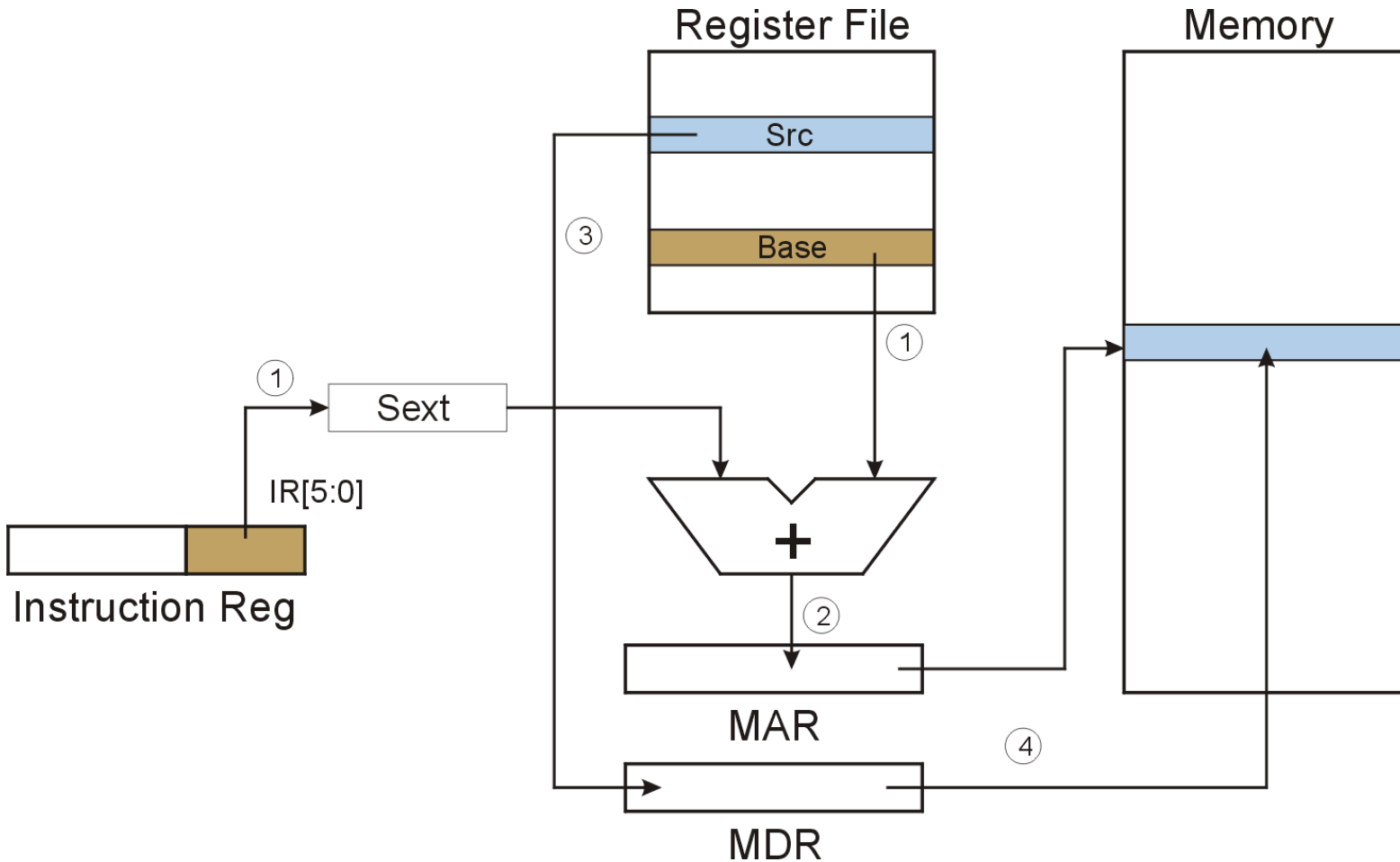
Assembly Ex:
LDR R4, R1, #1

LDR (Base+Offset)



STR (Base+Offset)

Assembly Ex:
STR R4, R1, #1



Load Effective Address

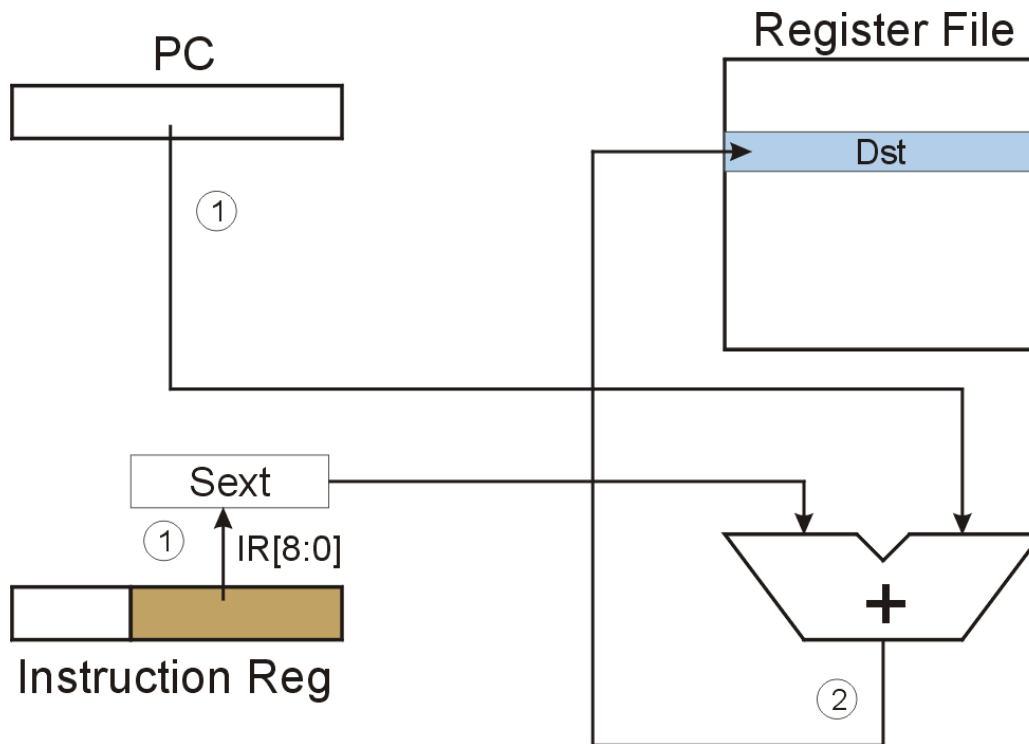
Computes address like PC-relative (PC plus signed offset) and **stores the result into a register.**

Note: The address is stored in the register, not the contents of the memory location.

```
LEA    R1, Begin  
LDR    R3, R1, #0
```

We can use the destination register as a pointer

LEA (Immediate)



Assembly Ex:
LEA R1, Lab1

Example (with RTL)

Address	Instruction	Comments
x30F6	1 1 1 <u>0 0 0 1 1 1 1 1 1 1 0 1</u>	$R1 \leftarrow PC - 3 = x30F4$
x30F7	0 0 0 <u>1 0 1 0 0 0 1 1 0 1 1 1 0</u>	$R2 \leftarrow R1 + 14 = x3102$
x30F8	0 0 1 <u>1 0 1 0 1 1 1 1 1 1 0 1 1</u>	$M[PC - 5] \leftarrow R2$ $M[x30F4] \leftarrow x3102$
x30F9	0 1 0 <u>1 0 1 0 0 1 0 1 0 0 0 0 0</u>	$R2 \leftarrow 0$
x30FA	0 0 0 <u>1 0 1 0 0 1 0 1 0 0 1 0 1</u>	$R2 \leftarrow R2 + 5 = 5$
x30FB	0 1 1 <u>1 0 1 0 0 0 1 0 0 1 1 1 0</u>	$M[R1+14] \leftarrow R2$ $M[x3102] \leftarrow 5$
x30FC	1 0 1 <u>0 0 1 1 1 1 1 1 1 0 1 1 1</u>	$R3 \leftarrow M[M[x30F4]]$ $R3 \leftarrow M[x3102]$ $R3 \leftarrow 5$

opcode

Example (in assembly)

<i>Address</i>	<i>Instruction</i>	<i>Comments</i>
x30F6	1 1 1 0 0 0 1 1 1 1 1 1 1 0 1	LEA R1, Lab2
x30F7	0 0 0 1 0 1 0 0 0 1 1 0 1 1 1 0	ADD R2, R1, #14
x30F8	0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1	ST R2, Lab2
x30F9	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	AND R2, R2, #0
x30FA	0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1	ADD R2, R2, #5
x30FB	0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0	LDR R2, R1, #14
x30FC	1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1	LDI R2, Lab2

LC3 Addressing Modes: Comparison

Instruction	Example	Destination	Source
NOT	NOT R2, R1	R2	R1
ADD / AND (imm)	ADD R3, R2, #7	R3	R2, #7
ADD /AND	ADD R3, R2, R1	R3	R2, R1

LD	LD R4, LABEL	R4	M[LABEL]
ST	ST R4, LABEL	M[LABEL]	R4
LDI	LDI R4, HERE	R4	M[M[HERE]]
STI	STI R4, HERE	M[M[HERE]]	R4
LDR	LDR R4, R2, #-5	R4	M[R2 - 5]
STR	STR R4, R2, #5	M[R2 + 5]	R4
LEA	LEA R4, TARGET	R4	address of TARGET

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

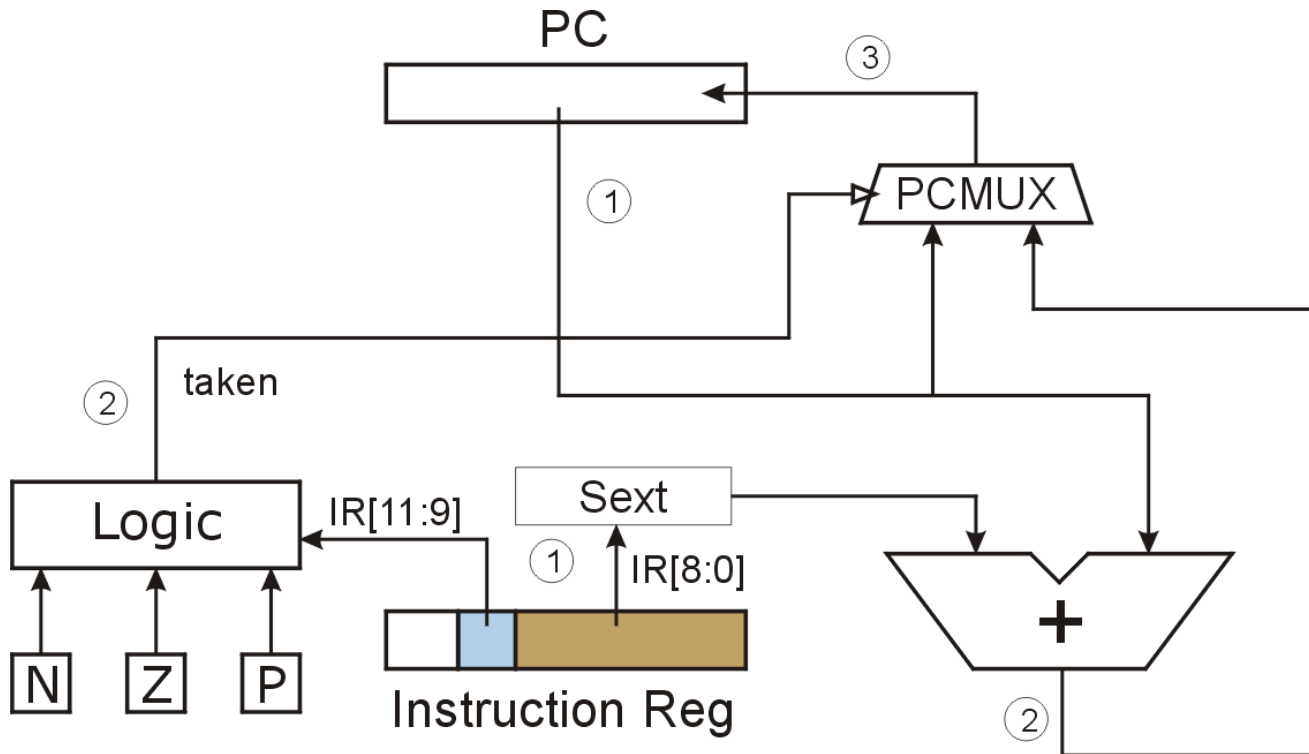
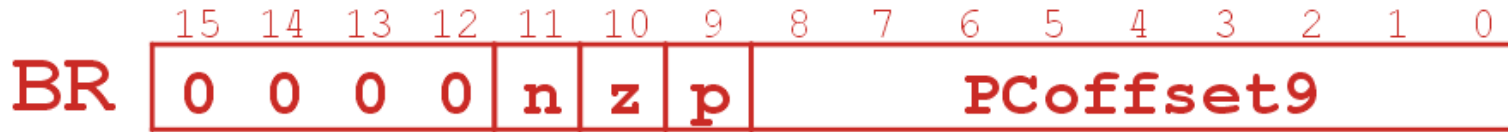
Branch specifies one or more condition codes.

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.

BR (PC-Relative)

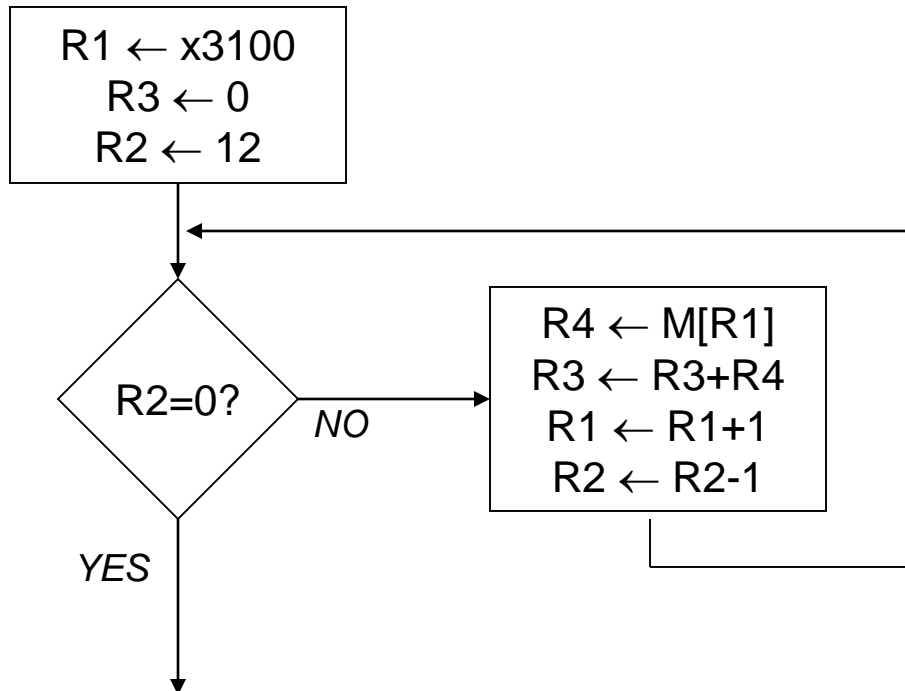


What happens if bits [11:9] are all zero? All one?

Using Branch Instructions

Compute sum of 12 integers.

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

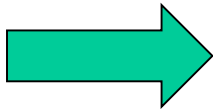


Sum of 4 integers

;Computes sum of integers
;R1: pointer, initialized to NUMS (x300C)
;R3: sum, initially cleared, accumulated here
;R2: down counter, initially holds number of numbers 4

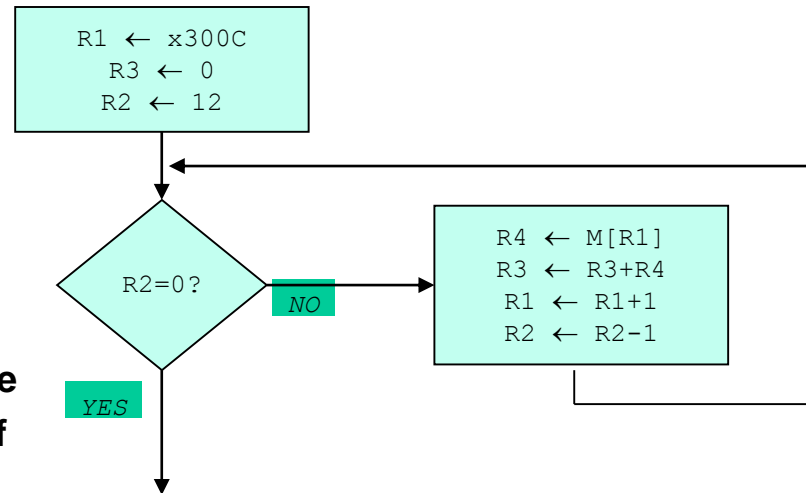
```
.ORIG    0x3000
```

.....



```
DONE    ST R3, SUM ;added  
        HALT
```

```
NUMS    .FILL 3  
        .FILL -4  
        .FILL 7  
        .FILL 3  
SUM     .BLKW 1  
        .END
```



```
LEA R1,NUMS  
AND R3,R3, #0  
AND R2,R2, #0  
ADD R2, R2, #4
```

```
LOOP    BRz  DONE  
        LDR R4,R1,#0  
        ADD R3,R3,R4  
        ADD R1,R1,#1  
        ADD R2,R2,#-1  
        BRnzp LOOP
```

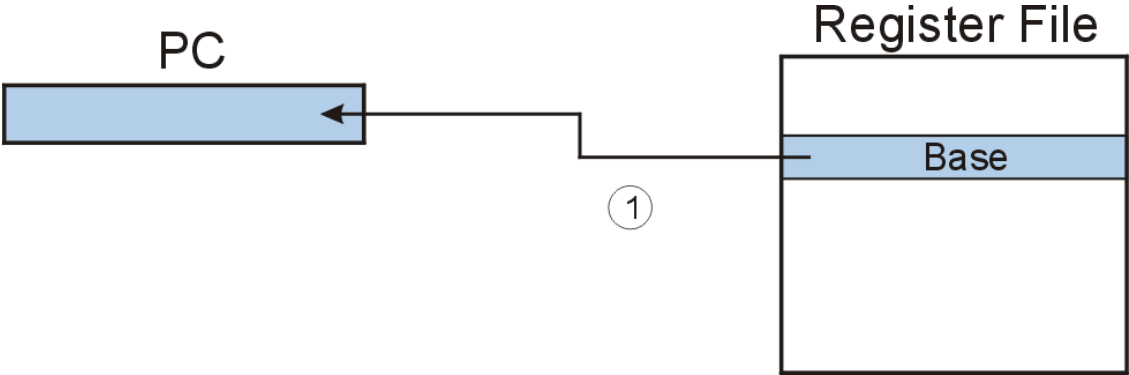
Sample Program

Address	Instruction	Comments
x3000	1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1	$R1 \leftarrow x3100$ (PC+0xFF)
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	$R3 \leftarrow 0$
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	$R2 \leftarrow 0$
x3003	0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0	$R2 \leftarrow 12$
x3004	0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1	If Z, goto x300A (PC+5)
x3005	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	Load next value to R4
x3006	0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1	Add to R3
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	Increment R1 (pointer)
X3008	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	Decrement R2 (counter)
x3009	0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0	Goto x3004 (PC-6)

JMP (Register)

Jump is an unconditional branch -- always taken.

- Target address is the contents of a register.
- Allows any target address.



TRAP



Calls a **service routine**, identified by 8-bit “trap vector.”

<i>vector</i>	<i>routine</i>
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.)

Another Example

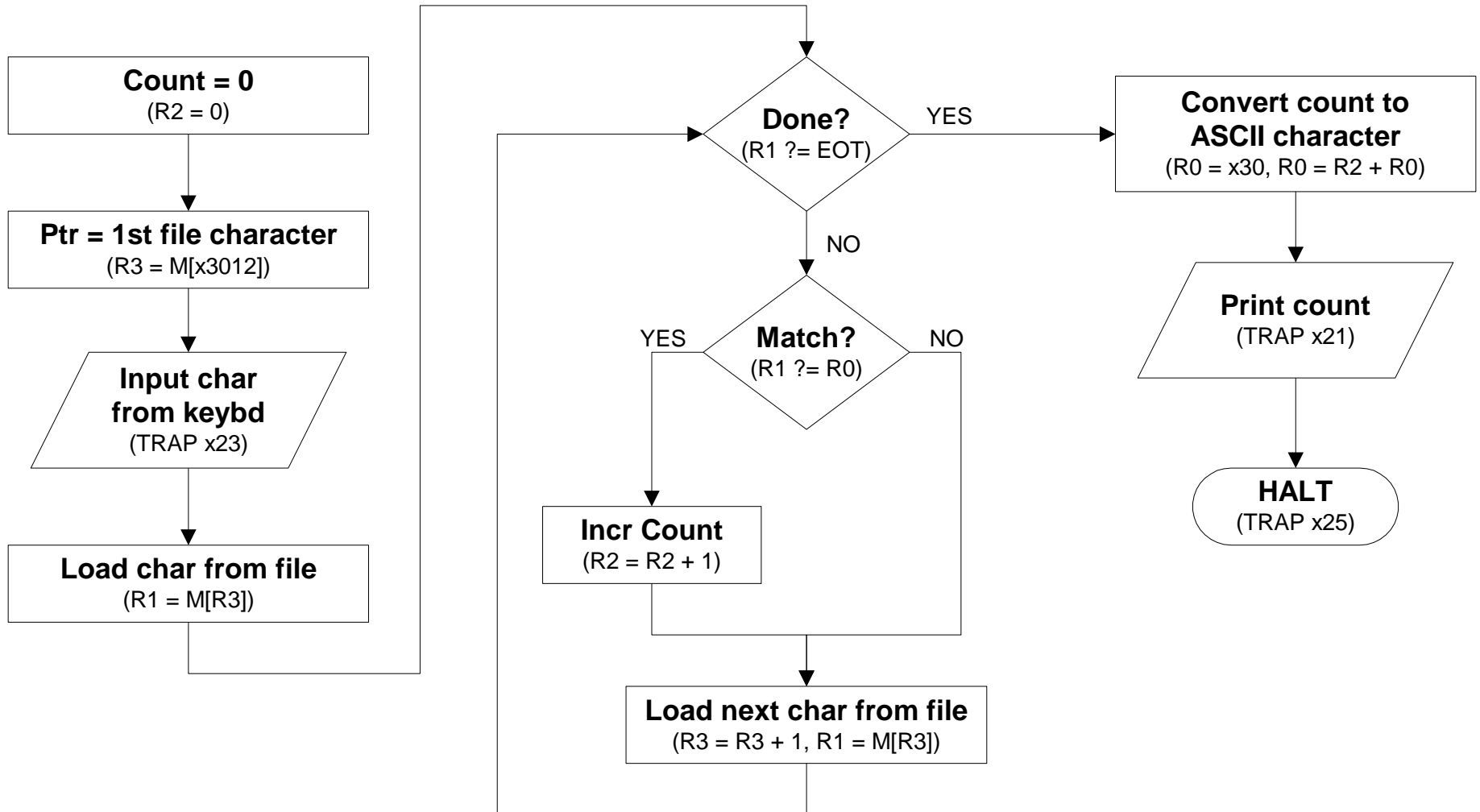
Count the occurrences of a character in a file

- **Program begins at location x3000**
- **Read character from keyboard**
- **Load each character from a “file”**
 - File is a sequence of memory locations
 - Starting address of file is stored in the memory location immediately after the program
- **If file character equals input character, increment counter**
- **End of file is indicated by a special ASCII value: EOT (x04)**
- **At the end, print the number of characters and halt**
(assume there will be less than 10 occurrences of the character)

A special character used to indicate the end of a sequence is often called a **sentinel.**

- Useful when you don't know ahead of time how many times to execute a loop.

Flow Chart



```

.ORIG x3000
AND R2,R2,#0 ; R2 is counter, initialize to 0
LD R3,PTR ; R3 is pointer to characters
TRAP x23 ; R0 gets character input
LDR R1,R3,#0 ; R1 gets the next character
;
; Test character for end of file
;
TEST ADD R4,R1,#-4 ; Test for EOT
BRz OUTPUT ; If done, prepare the output
;
; Test character for match. If a match, increment count.
;
NOT R1,R1
ADD R1,R1,R0 ; If match, R1 = xFFFF
NOT R1,R1 ; If match, R1 = x0000
BRnp GETCHAR ; no match, do not increment
ADD R2,R2,#1
;

```

```

; Get next character from the file
;
GETCHAR ADD R3,R3,#1 ; Increment the pointer
LDR R1,R3,#0 ; R1 gets the next character to
test
BRnzp TEST
;
; Output the count.
;
OUTPUT LD R0,ASCII ; Load the ASCII template
ADD R0,R0,R2 ; Convert binary to ASCII
TRAP x21 ; ASCII code in R0 is displayed
TRAP x25 ; Halt machine
;
; Storage for pointer and ASCII template
;
ASCII .FILL x0030
PTR .FILL x3015
.END

```

Program (1 of 2)

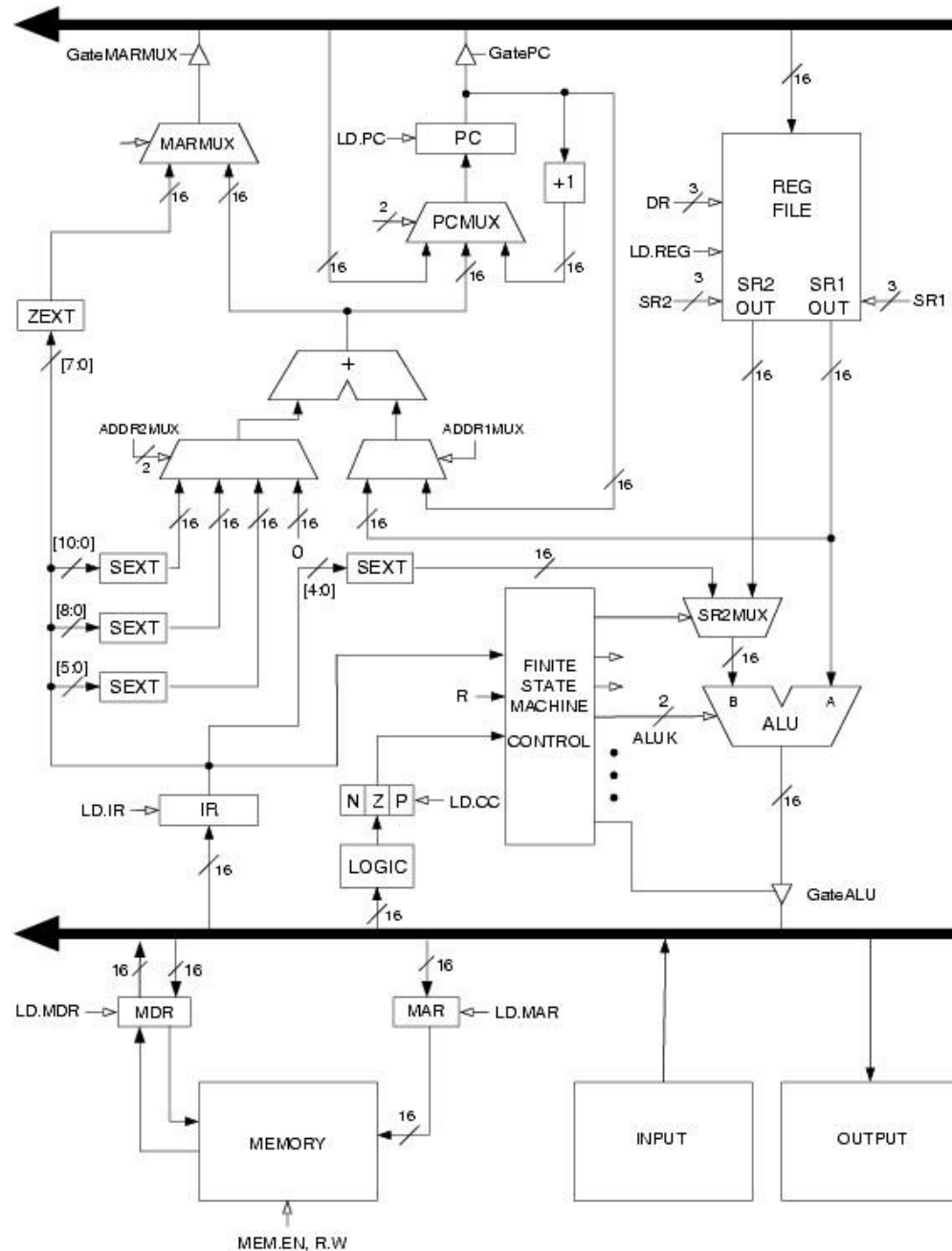
<i>Address</i>	<i>Instruction</i>	<i>Comments</i>
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	$R2 \leftarrow 0$ (counter)
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0	$R3 \leftarrow M[x3102]$ (ptr)
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	Input to R0 (TRAP x23)
x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	$R1 \leftarrow M[R3]$
x3004	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0	$R4 \leftarrow R1 - 4$ (EOT)
x3005	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	If Z, goto x300E
x3006	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	$R1 \leftarrow \text{NOT } R1$
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	$R1 \leftarrow R1 + 1$
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	$R1 \leftarrow R1 + R0$
x3009	0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1	If N or P, goto x300B

Program (2 of 2)

<i>Address</i>	<i>Instruction</i>	<i>Comments</i>
x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	$R2 \leftarrow R2 + 1$
x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	$R3 \leftarrow R3 + 1$
x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	$R1 \leftarrow M[R3]$
x300D	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0	<i>Goto x3004</i>
x300E	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0	$R0 \leftarrow M[x3013]$
x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0	$R0 \leftarrow R0 + R2$
x3010	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1	<i>Print R0 (TRAP x21)</i>
x3011	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	<i>HALT (TRAP x25)</i>
X3012	Starting Address of File	
x3013	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	<i>ASCII x30 ('0')</i>

LC-3 Data Path Revisited

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



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).**
- **Output goes to bus.**
 - **used by condition code logic, register file, memory**

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

More details later.

Multiplexer (MUX): selects data from multiple sources

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.