

ECE290 Fall 2012

Lecture 20

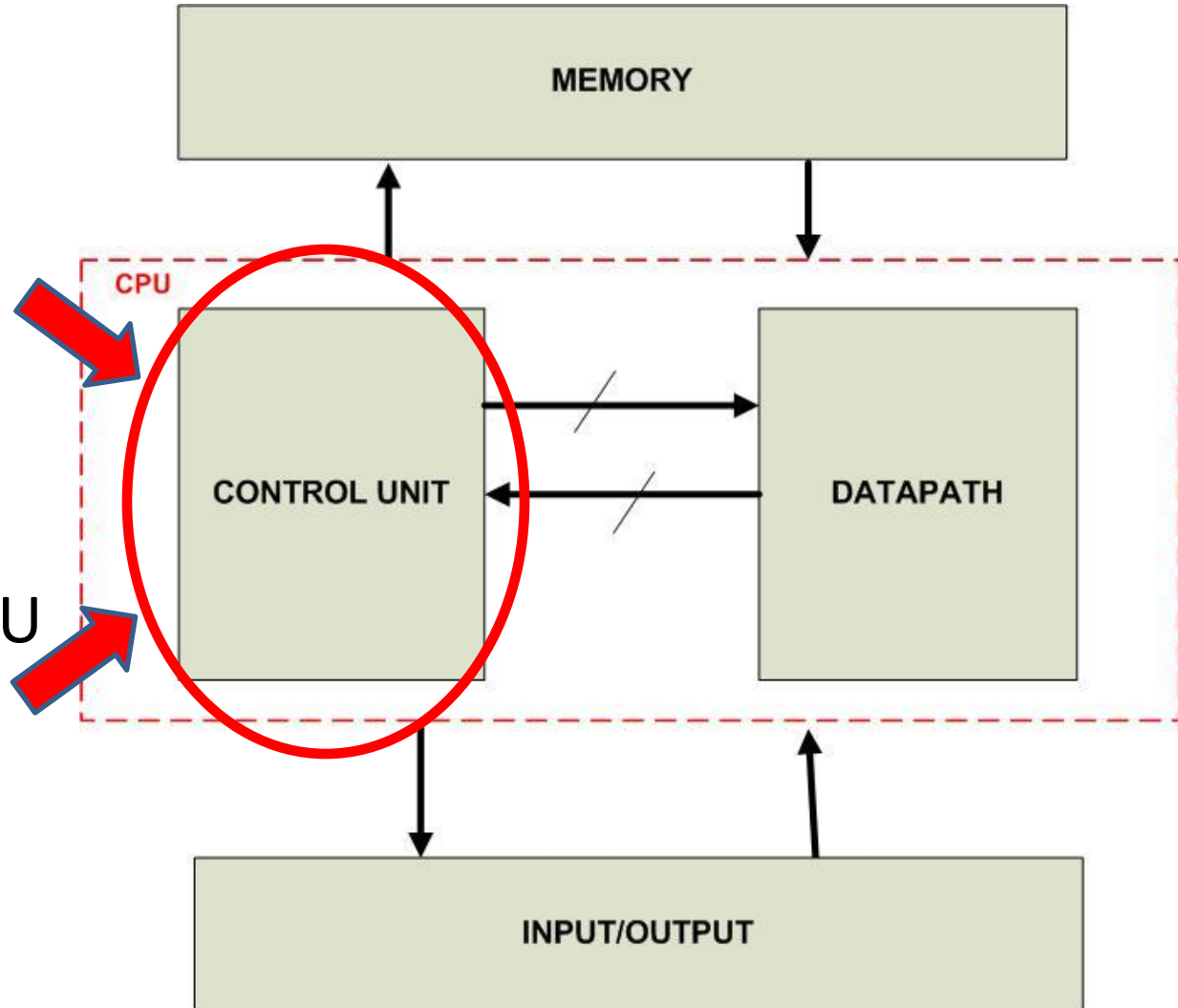
Dr. Zbigniew Kalbarczyk

Today

- Central Processing Unit
 - LC-3 Architecture
 - A program example
 - Assembly instructions
 - Binary instructions

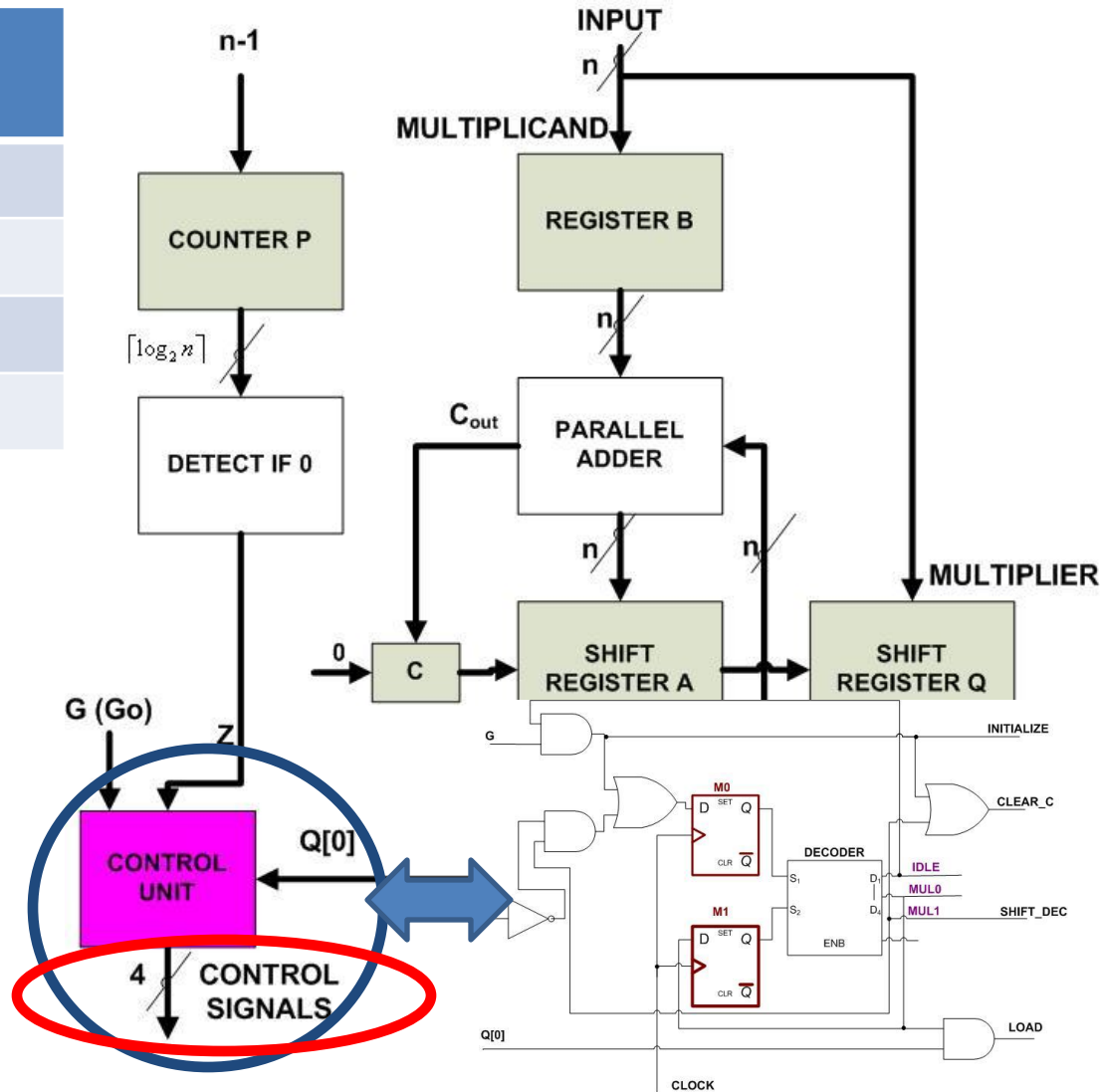
Components of a Simple Computer

- **Before:** Focus on designing a control unit of a CPU in a simple computer
- **Now:** Focus on understanding a control unit of a CPU in a sophisticated computer



Before: Binary Multiplier: Data Path & Control Unit

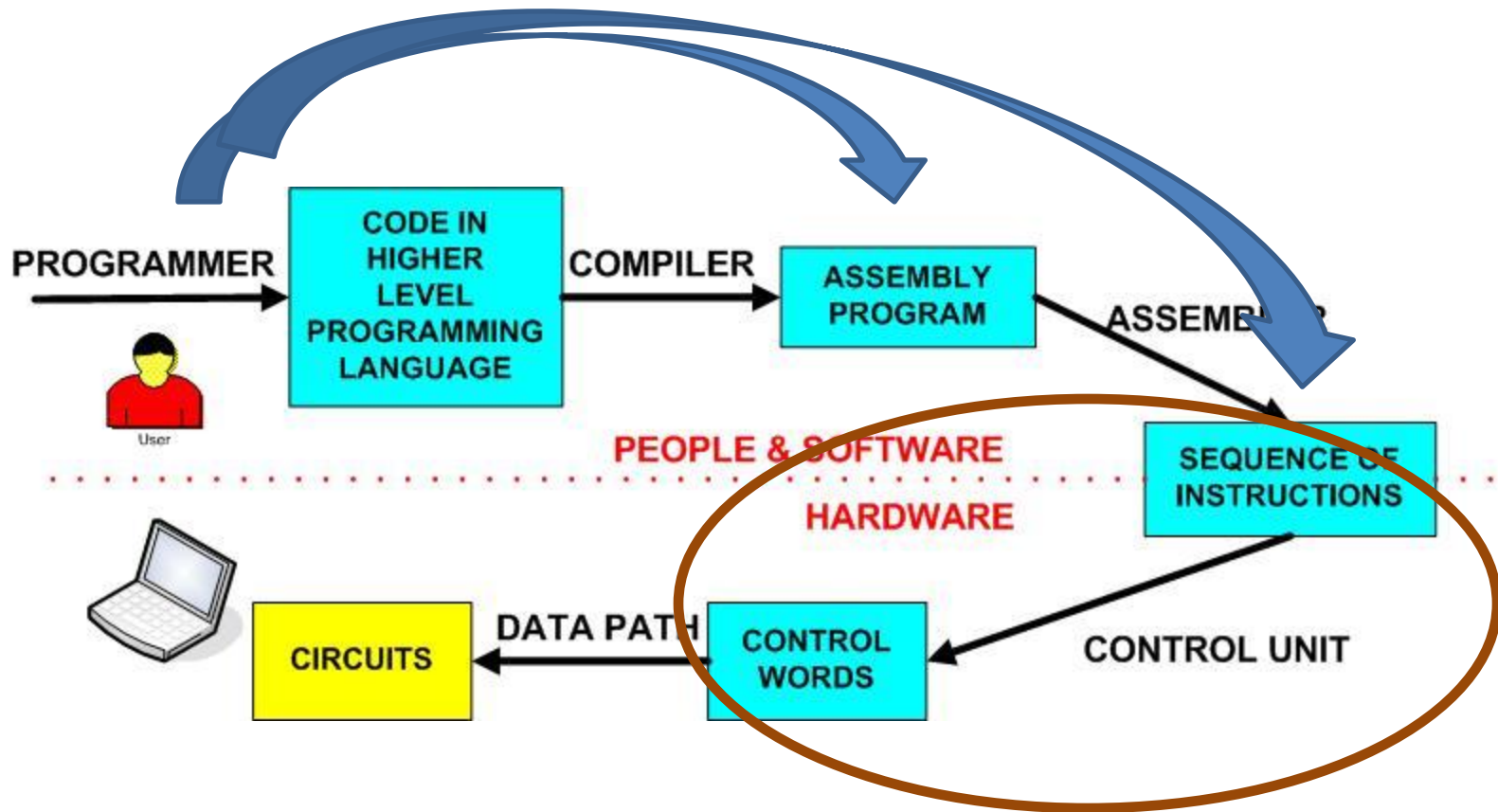
Control Signal Name	Boolean Expression
Initialize	$IDLE * G$
Load_AC	$MUL0 * Q[0]$
Shift_dec	$MUL1$
Clear_C	$MUL1 + IDLE * G$



Now: Little Computer (LC) - 3

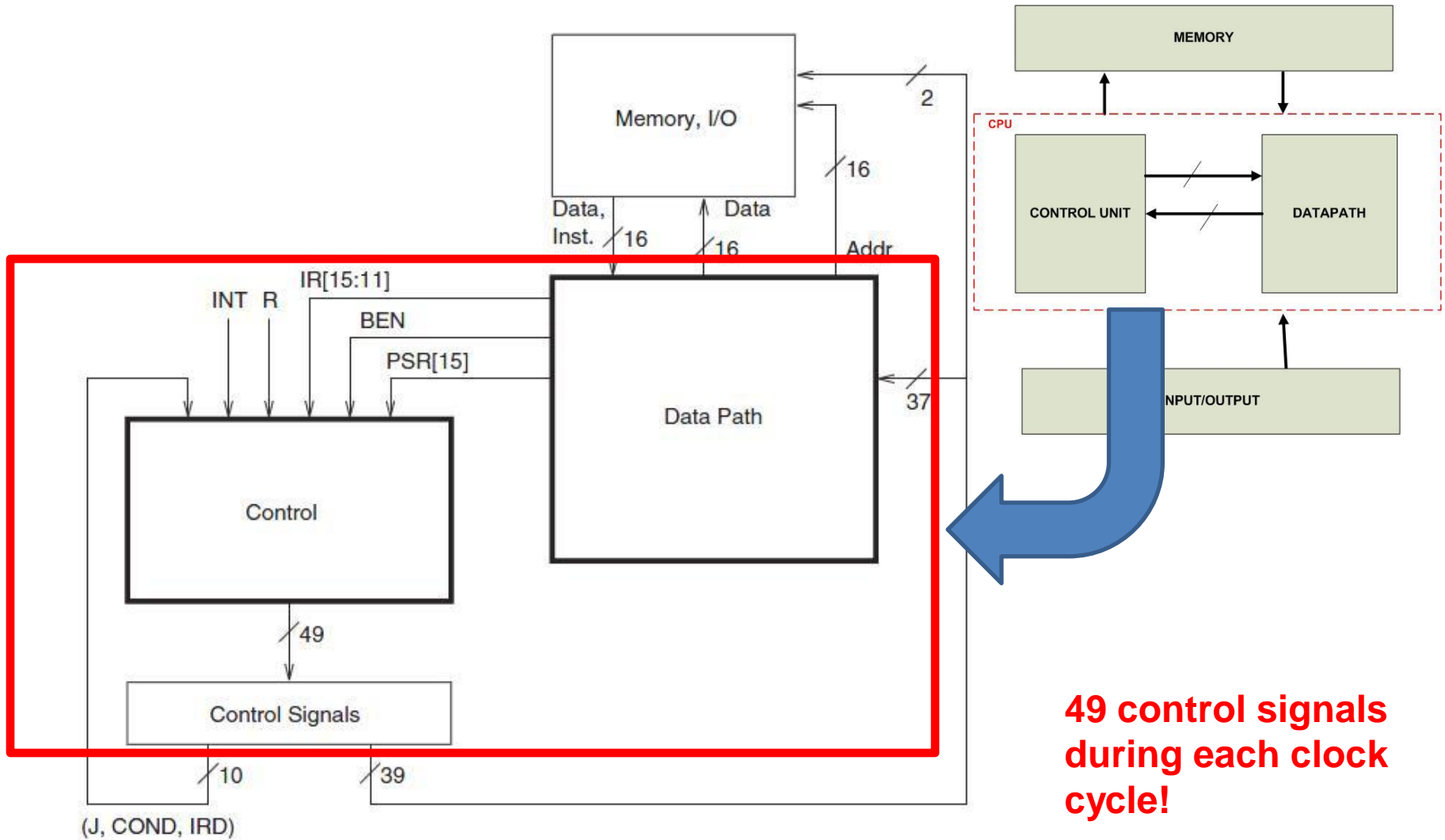
- The LC-3 serves as a transition from a single purpose (binary multiplier) to a multi-purpose real CPU
- The LC-3 has all the important characteristics of microprocessors
 - The Intel 8080 (in the first IBM PC ~ 1981)
 - The Motorola 68000 (in the Macintosh ~ 1984)
 - The Pentium IV (in PCs ~ 2003)
 -

Interfacing LC-3



We study the process of translating a sequence of instructions to control words

LC-3: Major Components



Signals from Data Path to Control

- **IR[15:11]** are the opcode bits
- **PSR[15]** is the bit [15] of the Processor Status Register, which indicates whether the current program is executing with supervisor or user privileges.
- **BEN** is to indicate whether or not a BR (conditional branch) should be taken.
- **INT** is to indicate that some external device of higher priority than the executing process requests service.
- **R** is to indicate the end of a memory operation.

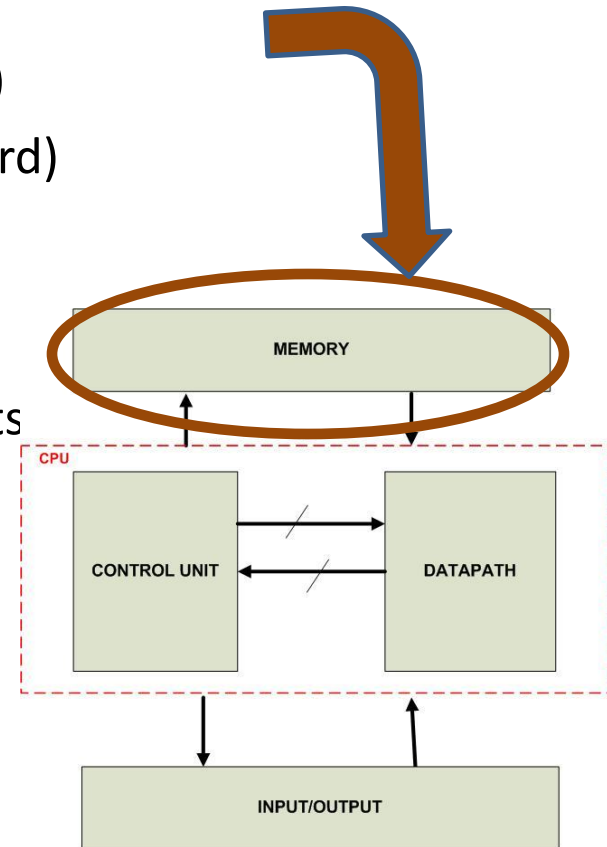
LC-3: Memory Description

- **Memory Unit:**

- $2^{16} \times 16$ RAM (64K x 2Bytes = 128KB)
 - Memory Address Register (MAR): 16 bits (64K)
 - Memory Data Register (MDR): 16 bits (one word)

- **Memory Units in Existing Processors**

- Intel's PENTIUM IV ~ MDR= 32 bits
- Sun's Sparc-V9 & Intel's Itanium ~ MDR= 64 bits
- Pagers, VCRs, cell phones ~ MDR = 8bits



LC-3: CPU Description

- **CPU:**

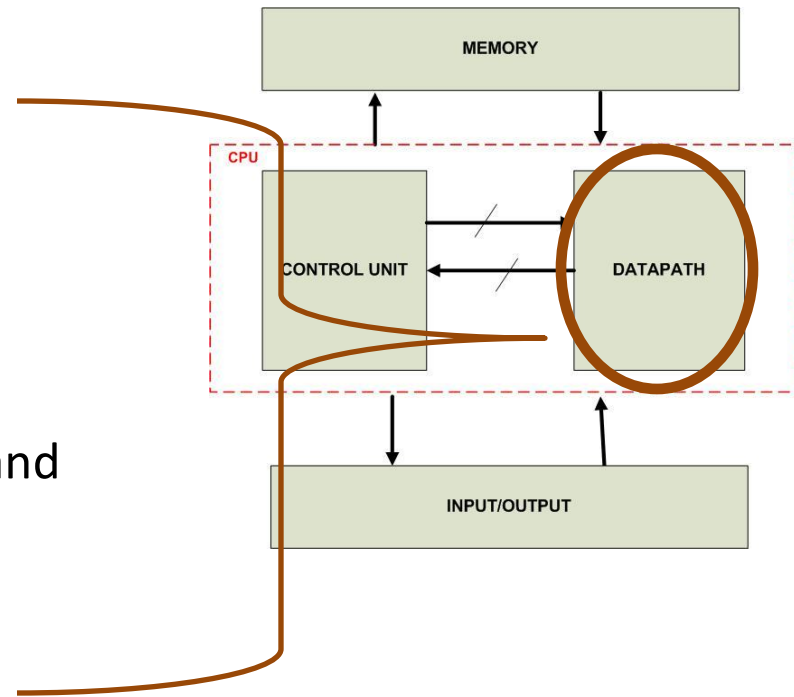
- **Data Path: Register File**

- General purpose registers R0-R7
 - Each register has 16 bits

- **Data Path: ALU**

- One arithmetic operation (ADD)
 - Two logical operations (bitwise AND and bitwise complement)

- Compare with Sun's Sparc-V9
 - 32 registers (64 bits ~ word)

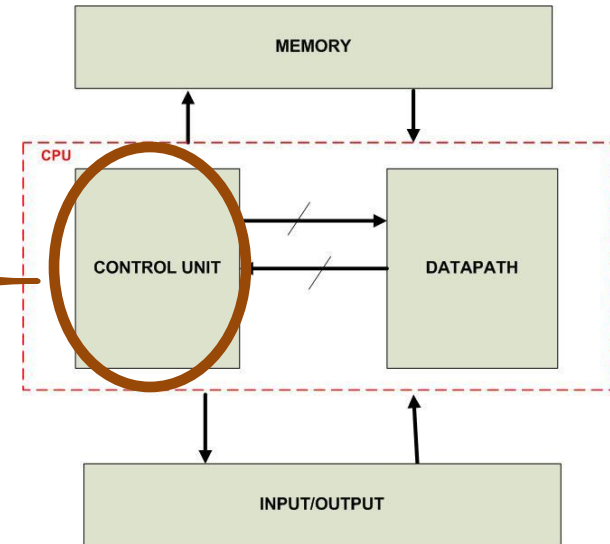


LC-3: CPU Description

- CPU

- Control Unit:

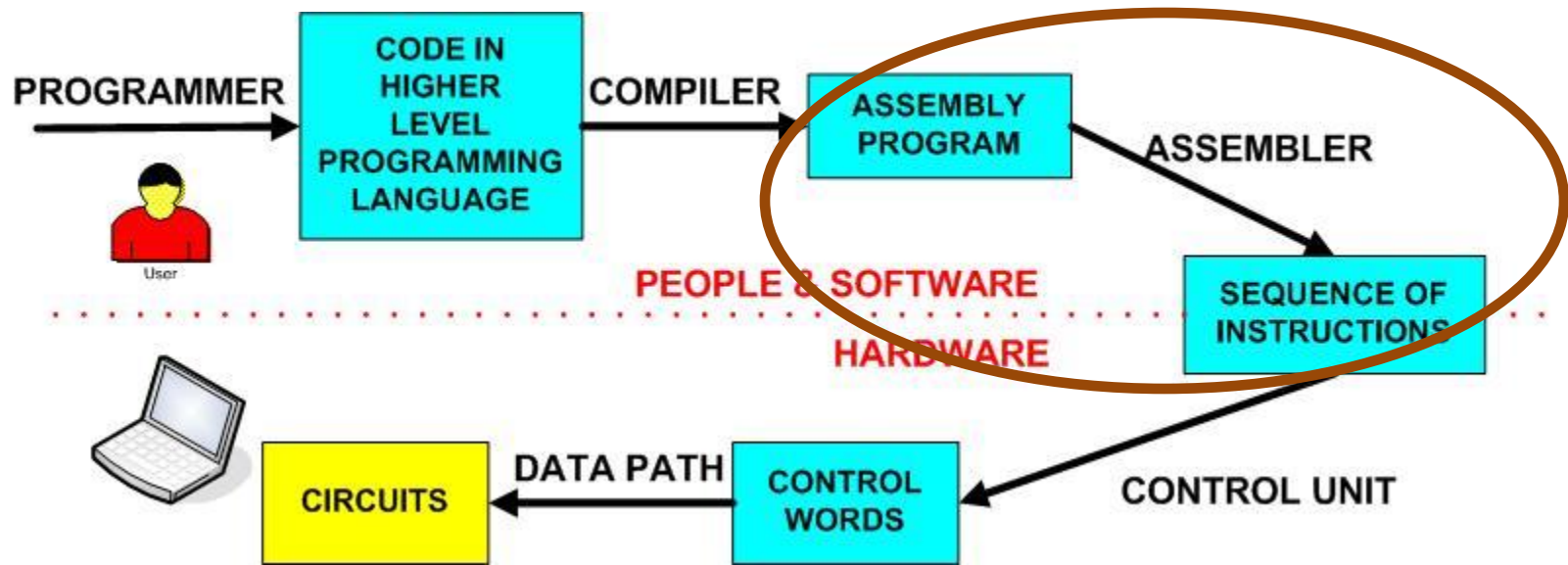
- Program Counter (PC) – 16 bit address of next instruction
 - Instruction register (IR) – 16 bit word from memory interpreted as instruction
 - Condition codes NZP – each N or Z or P indicates whether the result was negative (100), zero (010) or positive (001)



Note: Some registers (e.g., PC, MAR) are not directly addressable by the end user but are used by instructions

LC-3: Instruction Set Architecture (ISA)

- LC-3 ISA is the complete specification of the interface between programs and the underlying computer architecture
- ISA is the information required to write machine language program



LC-3 ISA: Instruction Set

- Instruction Set:
 - a set of instructions the LC-3 can execute
- Instruction Format:
 - binary representation of instructions (2 bytes ~ word)
- Instruction List: (There are 16 opcodes)
 - **Operations**
 - **Operate:** ADD, AND, NOT
 - **Data movement:** LD, LDI, LDR, LEA, ST, STI, STR
 - **Control:** BR, JMP, JSR, JSRR, RET, RTI, TRAP
 - **Data types:** 16 bit 2's complement numbers

LC-3 ISA: Addressing Modes

- Where are the operands?
 - register, memory, immediate
- How is location of an operand specified?
 - Non-memory addresses
 - Register,
 - Immediate (literal)
 - Memory addresses
 - Program counter relative,
 - Base + offset,
 - Indirect

LC-3 ISA: Li

• Operations

– Operate: ADD^+

– Data move: JMP

– Control: BI

ADD^+

ADD^+

AND^+

AND^+

ABR

JMP

JSR

JSRR

LD^+

LDI^+

LDR^+

LEA^+

NOT^+

RET

ST

STI

STR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0001				DR				SR1		0		00			SR2
0001				DR				SR1		1		imm5			
0101				DR				SR1		0		00			SR2
0101				DR				SR1		1		imm5			
0000			n	z	p										PCoffset9
1100				000				BaseR							000000
0100				1											PCoffset11
0100				0		00		BaseR							000000
0010				DR											PCoffset9
1010				DR											PCoffset9
0110				DR				BaseR							offset6
1110				DR											PCoffset9
1001				DR				SR							111111
1100				000				111							000000
0011				SR											PCoffset9
1011				SR											PCoffset9
0111				SR				BaseR							offset6

$\text{DR} \leftarrow \text{SR1} + \text{SR2};$
set NZP

$\text{DR} \leftarrow \text{SR1} + \text{SEXT}(\text{imm5});$
set NZP

$\text{DR} \leftarrow \text{SR1 AND SR2};$
set NZP

$\text{DR} \leftarrow \text{SR1 AND SEXT}(\text{imm5});$
set NZP

IF $((n \cdot N) + (z \cdot Z) + (p \cdot P))$
THEN $\text{PC} \leftarrow \text{PC} + \text{SEXT}(\text{PCoffset9})$

$\text{PC} \leftarrow \text{BaseR}$

$\text{R7} \leftarrow \text{PC}$
 $\text{PC} \leftarrow \text{PC} + \text{SEXT}(\text{PCoffset11})$

$\text{R7} \leftarrow \text{PC}$
 $\text{PC} \leftarrow \text{BaseR}$

$\text{DR} \leftarrow \text{M}[\text{PC} + \text{SEXT}(\text{PCoffset9})];$
Set NZP

$\text{DR} \leftarrow \text{M}[\text{M}[\text{PC} + \text{SEXT}(\text{PCoffset9})]];$
Set NZP

$\text{DR} \leftarrow \text{M}[\text{BaseR} + \text{SEXT}(\text{offset6})];$
Set NZP

$\text{DR} \leftarrow \text{PC} + \text{SEXT}(\text{PCoffset9});$
Set NZP

$\text{DR} \leftarrow \text{NOT}(\text{SR});$
Set NZP

$\text{PC} \leftarrow \text{R7}$

$\text{M}[\text{PC} + \text{SEXT}(\text{PCoffset9})] \leftarrow \text{SR}$

$\text{M}[\text{M}[\text{PC} + \text{SEXT}(\text{PCoffset9})]] \leftarrow \text{SR}$

$\text{M}[\text{BaseR} + \text{SEXT}(\text{offset6})] \leftarrow \text{SR}$

superscript "+" denotes instructions that update the condition bits NZP

Notation:

DR .. Destination regis

SR ...Source register (

Imm5 ... A 5 bit immed

BaseR ...R0-R7

offset6 ... BaseR + off

PCoffset9 ...PC+[-256

PCoffset11...PC+[-102

SEXT(A) ...sign-exten

ZEXT(A) ...zero-exten

Setcc() ... indicates the


LC-3: Using Operate Instructions

- **How do we execute other operations than ADD, AND and NOT?**
- **Copy:** $R1 \leftarrow R2$
 - Strategy: $R1 \leftarrow R2 + 0$ (immediate)
- **Clear:** $R1 \leftarrow 0$
 - Strategy: $R1 \leftarrow R1 \text{ AND } 0$ (immediate)
- **Subtract:** $R1 \leftarrow R2 - R3$
 - Strategy 1: $R1 \leftarrow \text{NOT } R3; R1 \leftarrow R1 + 1; R1 \leftarrow R2 + R1$
 - $R1 \leftarrow R2$ and 2's complement of $R3$
 - Strategy 2: if 2nd operand VAL is small then $R1 \leftarrow R2 + \text{VAL}$ (immediate)

LC-3: Using Operate Instructions

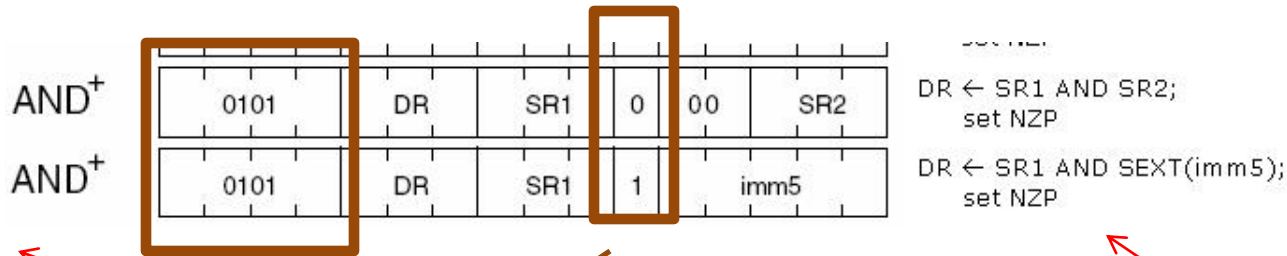
- **OR:** $R1 \leftarrow R2 \text{ OR } R3$
 - Strategy:

Register Operation	Boolean Operation
$R4 \leftarrow \text{NOT } R2$	$R4 = R2'$
$R5 \leftarrow \text{NOT } R3$	$R5 = R3'$
$R1 \leftarrow R4 \text{ AND } R5$	$R1 = R2' \text{ AND } R3'$
$R1 \leftarrow \text{NOT } R1$	$R1 = (R2' \text{ AND } R3')' = R2 \text{ OR } R3$



Used Only AND and NOT Operations!

LC-3: AND Instructions



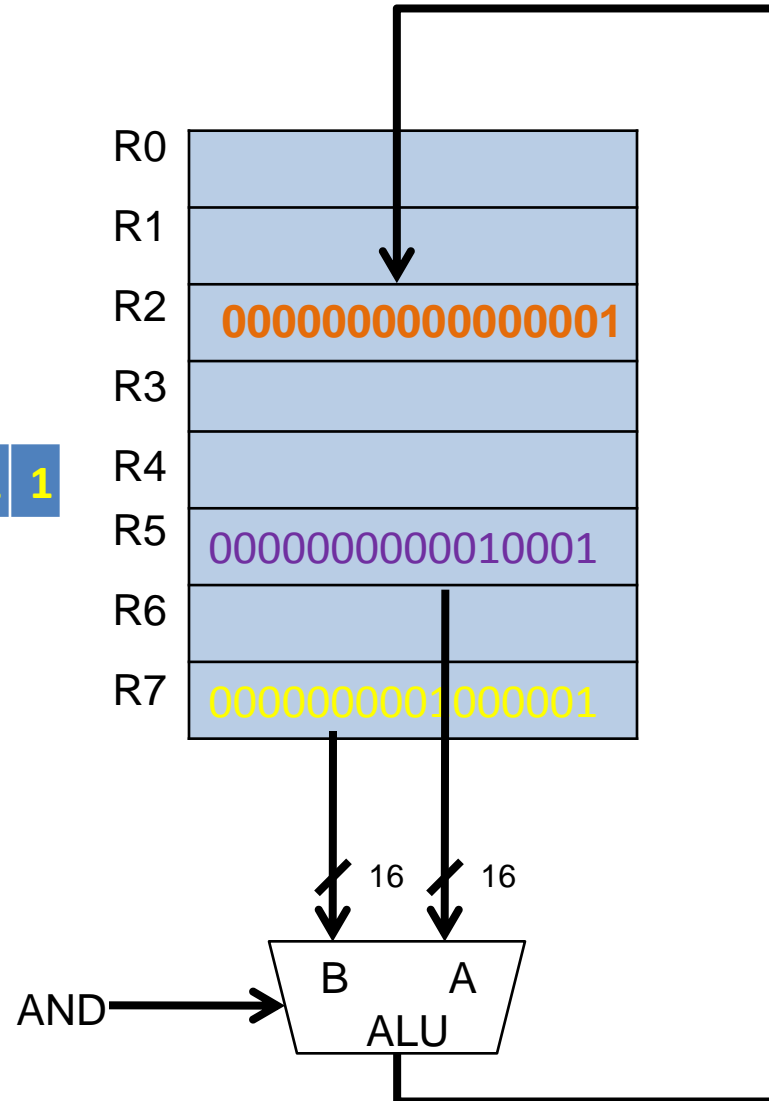
- OPCODE = 0101
- If bit[5] is zero
 - DR ← SR1 AND SR2 ~ 'register value'
- Else
 - DR ← SR1 AND SEXT(imm5) ~ 'immediate'
- Setcc() (means set N, Z and P based on in DR)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
ADD ⁺	0001						DR	SR1		0	00					SR2	DR ← SR1 + SR2; set NZP
ADD ⁺	0001						DR		SR1	1						imm5	DR ← SR1 + SEXT(imm5); set NZP
AND ⁺	0101						DR		SR1		0	00				SR2	DR ← SR1 AND SR2; set NZP
AND ⁺	0101						DR		SR1	1						imm5	DR ← SR1 AND SEXT(imm5); set NZP
BR	0000			n													IF ((nN)+(zZ)+(pP)) THEN PC ← PC + SEXT(PCoffset9)
JMP	1100																PC ← BaseR
JSR	0100																R7 ← PC PC ← PC + SEXT(PCoffset11)
JSRR	0100																R7 ← PC PC ← BaseR
LD ⁺	0010																DR ← M[PC + SEXT(PCoffset9)]; Set NZP
LDI ⁺	1010																DR ← M[PC + SEXT(PCoffset9)]; Set NZP
LDR ⁺	0110																DR ← M[BaseR + SEXT(offset6)]; Set NZP
LEA ⁺	1110																DR ← PC + SEXT(PCoffset9); Set NZP
NOT ⁺	1001																DR ← NOT(SR); Set NZP
RET	1100																PC ← R7
ST	0011																M[PC + SEXT(PCoffset9)] ← SR
STI	1011																M[M[PC + SEXT(PCoffset9)]] ← SR
STR	0111																M[BaseR + SEXT(offset6)] ← SR

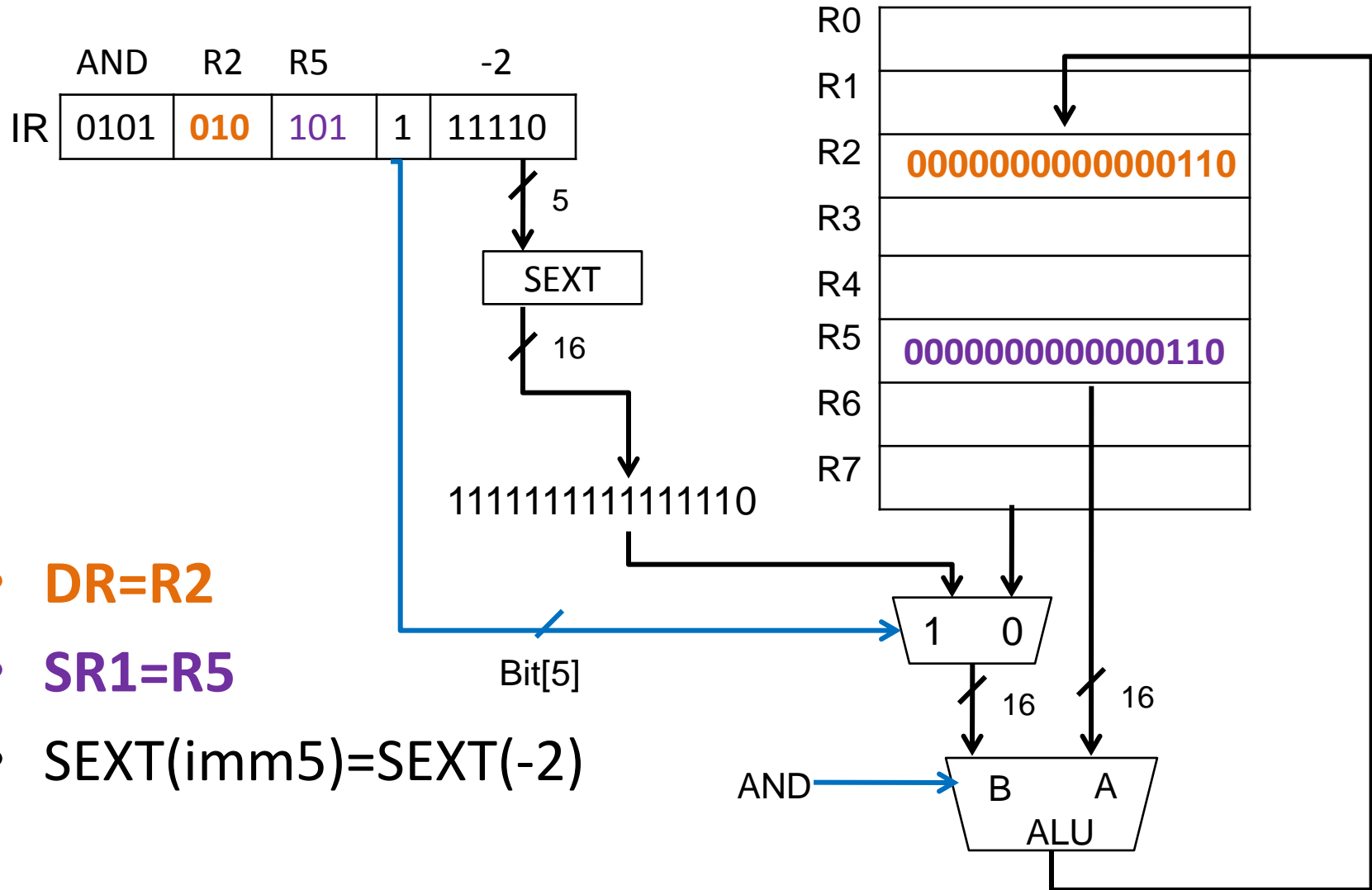
Example: $R2 \leftarrow R5 \text{ AND } R7$

- **DR=R2**
- **SR1=R5**
- **SR2 = R7**

IR 0 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1



Example: $R2 \leftarrow R5 \text{ AND SEXT}(-2)$

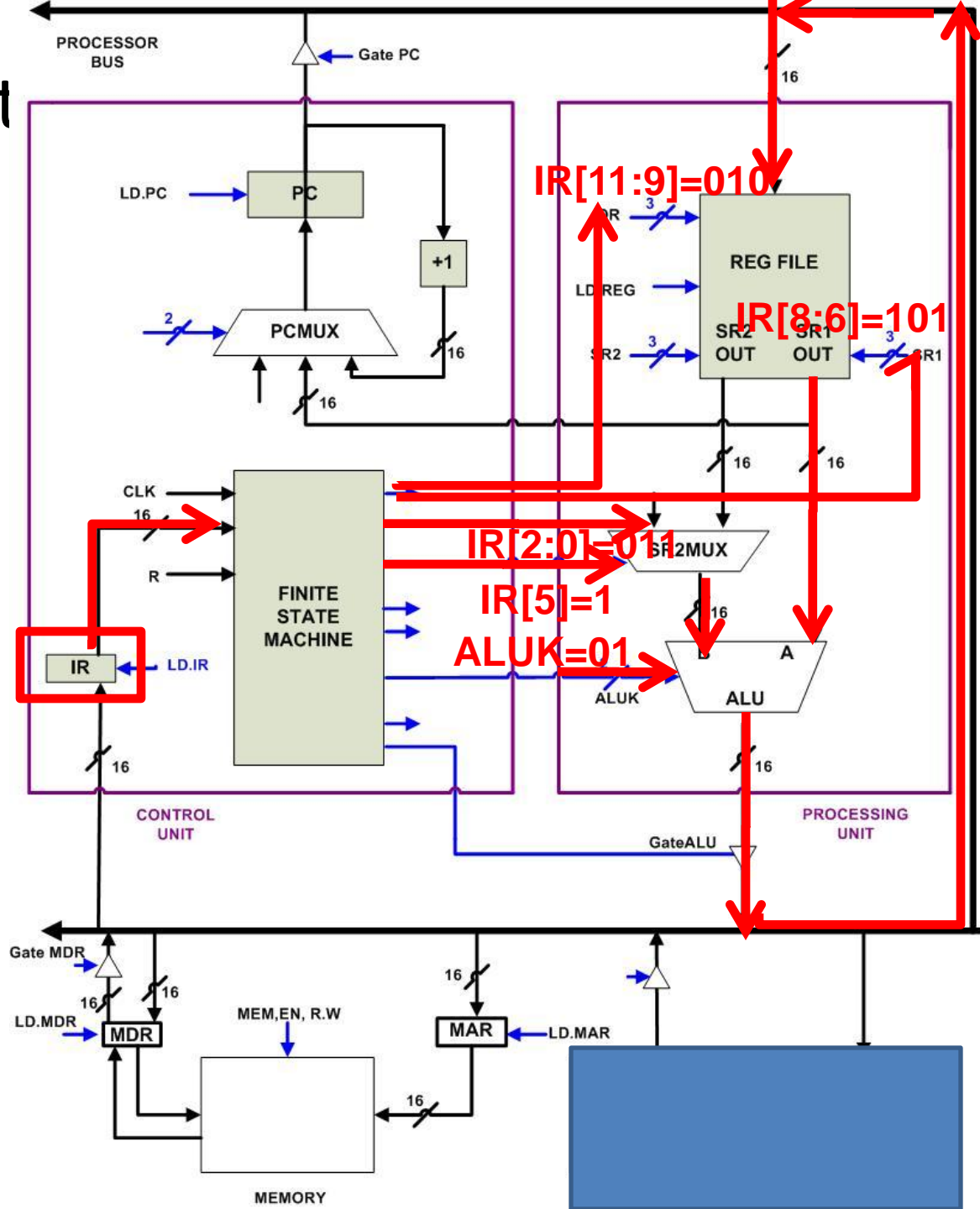


Example

- Initial Values
 - $R4 = \text{x4444}$, $R6 = \text{x0876}$
- LC-3 Instruction:
 - Address = x3101 , Instruction: `ADD R4, R6, #(-5)`
- **Give binary encoding of the LC-3 instruction:**
 - $(\text{ADD}, \text{DR}, \text{SR1}, 1, \text{imm5}) = 0001\ 100\ 110\ 1\ 11011$
- **Give the value of R4 after execution of the instruction:**
 - $(\text{x0876} + (-5)) = \text{x0871}$

LC-3 Archit

- Blue lines – control signals
- Black lines - data
- Example:
 - AND operation
 - $R2 \leftarrow R5 \text{ AND } \text{SEXT}(3)$











A Program Example

- Sum the numbers from 3 down to 1
- Store the result in memory location whose address is contained in R7


[illegible]

A Program Example: Register Status

- R1 – running sum, R2 - # to be added,
- R7 – ptr to memory

Address	Opcode	Operands	R1	R2	Comments
3000	AND	R1, R1, #0	0	X	% CLEAR R1
3001	ADD	R2, R1, #3	0	3	% LOAD 3 INTO R2
3002	ADD	R1, R1, R2	3 	3 	% ADD CURRENT # TO RUNNING SUM (R1)
3003	ADD	R2, R2, #-1	3 	2 	%DECREMENT # TO BE ADDED
3004	BRp	#-3	3 	2 	% IF R2>0 THEN REPEAT
3005	STR	R1, R7, #0			%STORE RESULT IN M[R7+0]

From Assembly Instructions to Binary Instructions



Address	Assembly Instruction	Register Transfer Language	Binary Instruction
3000	AND R1, R1, #0	PC \leftarrow PC+1; R1 \leftarrow 0	0101 001 001 1 00000
3001	ADD R2, R1, #3	PC \leftarrow PC+1; R2 \leftarrow R1 + #3	0001 010 001 1 00011
3002	ADD R1, R1, R2	PC \leftarrow PC+1; R1 \leftarrow R1 + R2	0001 001 001 0 00010
3003	ADD R2, R2, #-1	PC \leftarrow PC+1; R2 \leftarrow R2 -1	0001 010 010 1 11111
3004	BRp #-3	(PC \leftarrow PC+1) PC \leftarrow PC + SEXT(-3)	0000 001 111 1 11101
3005	STR R1, R7, #0	PC \leftarrow PC+1; M[R7+0] \leftarrow R1	0111 001 111 0 00000