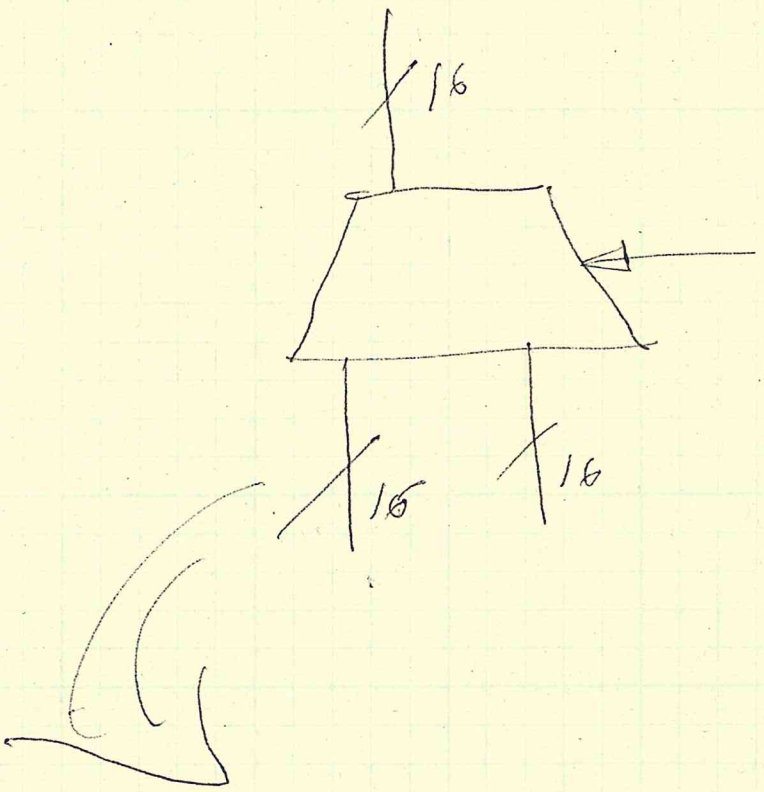
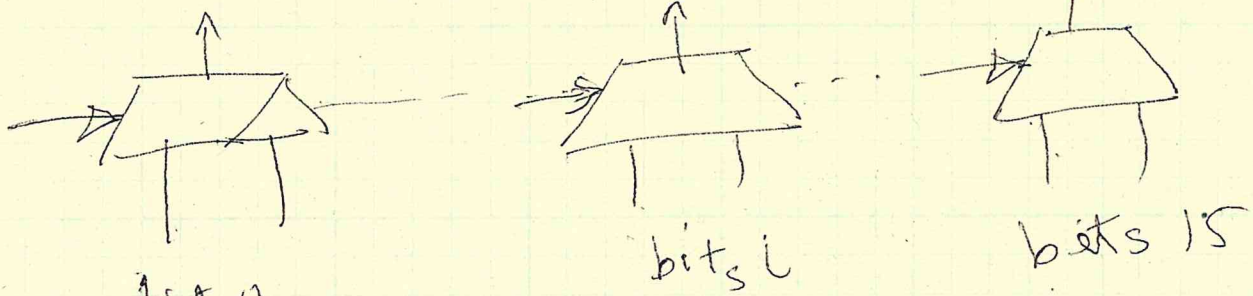
# 8. Instruction Set Architecture – The LC-3 (Chapter 5)

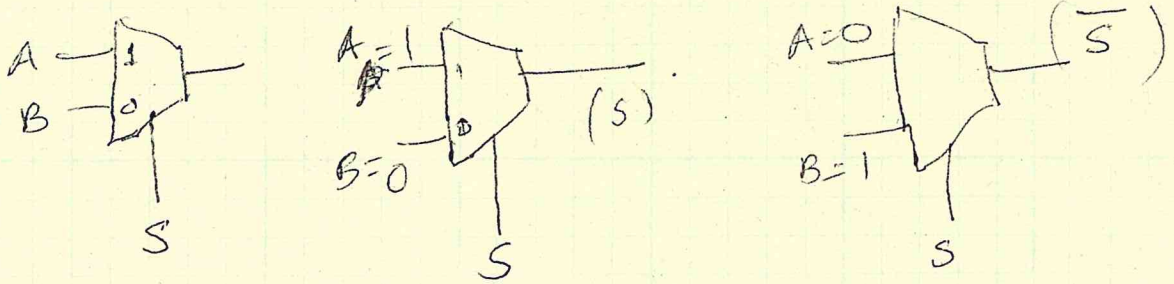
September 26, 2018

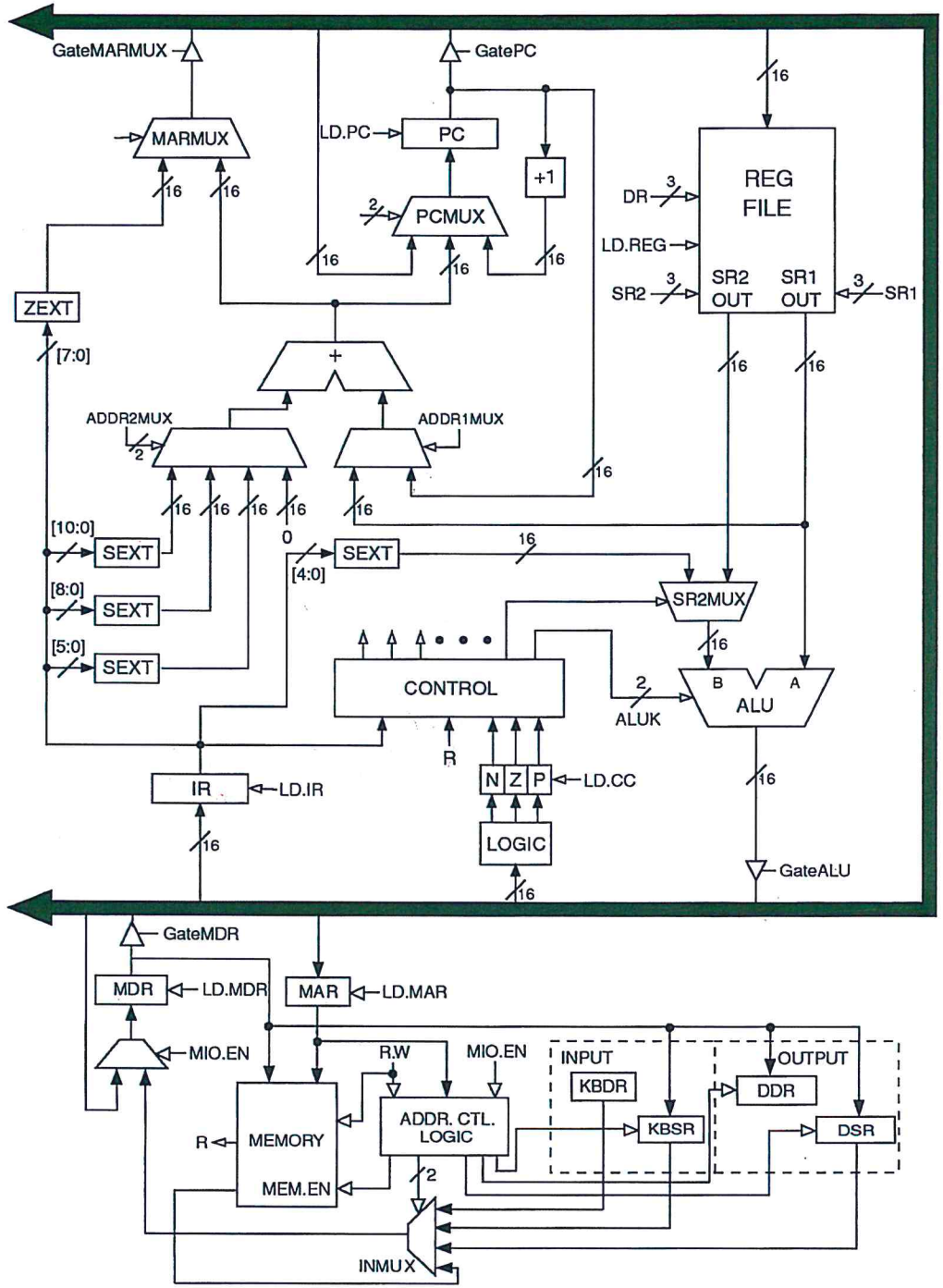
- Review
  - The von Neumann (stored program) computer
  - LC-3 data path
- Instruction processing
  - Example instructions

# MUX (SELECTOR)

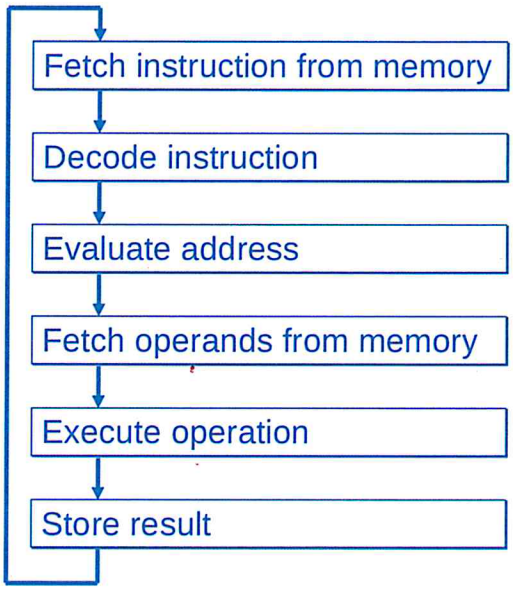


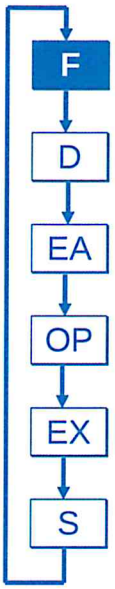

GateMARMUX
GatePC
MARMUX
LD.PC
PC
+1
PCMUX
REG FILE
DR
LD.REG
SR2
SR1
SR2 OUT
SR1 OUT
ZEXT
[7:0]
+
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
SEXT
[4:0]
SR2MUX
CONTROL
ALU
ALUK
R
IR
LD.IR
N
Z
P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
R.W
INPUT
KBDR
KBSR
OUTPUT
DDR
DSR
MEMORY
MEM.EN
ADDR. CTL. LOGIC
INMUX

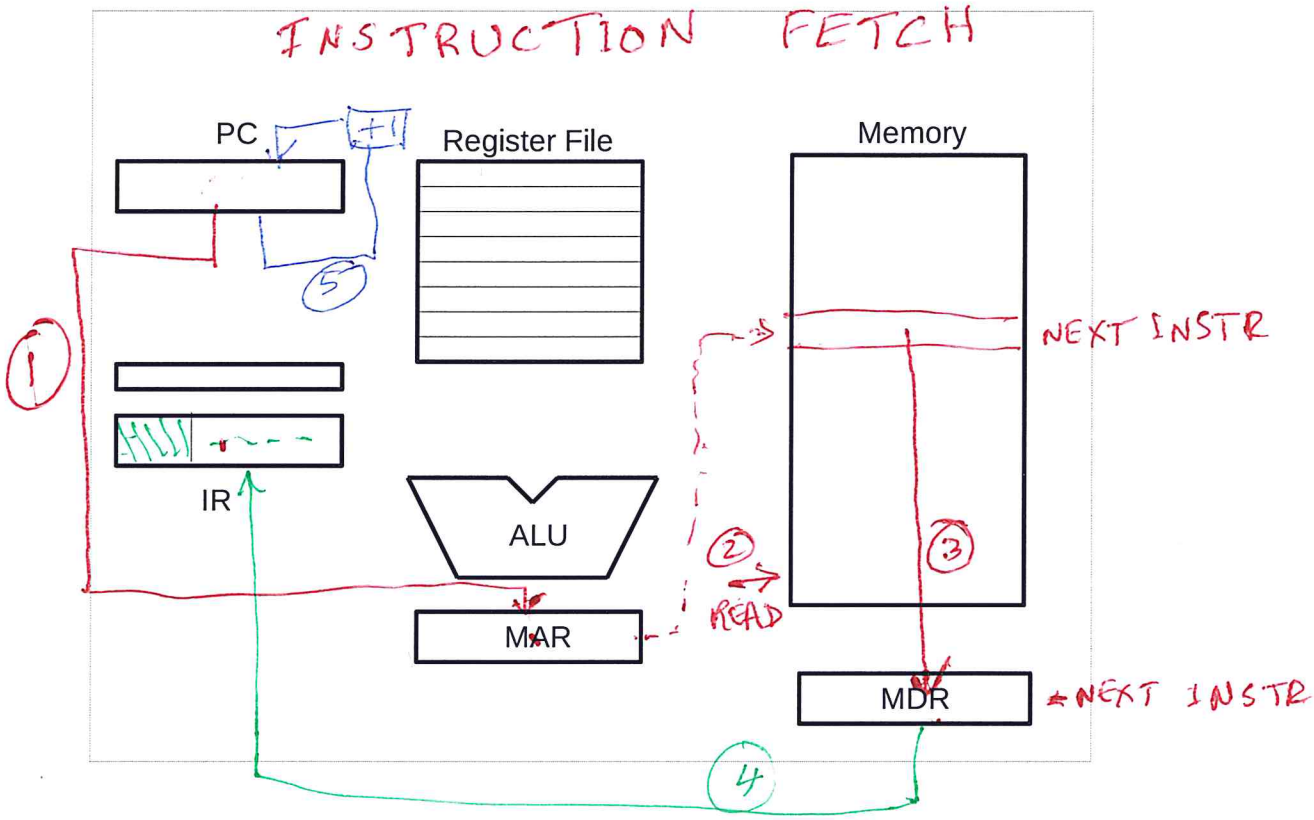
## Instruction Processing




ADDRESS IN PC

## Instruction Processing: FETCH




INSTRUCTION FETCH
PC
+1
Register File
Memory
NEXT INSTR
IR
ALU
MAR
READ
MDR
NEXT INSTR
1
2
3
4
5

## Instruction Processing: DECODE

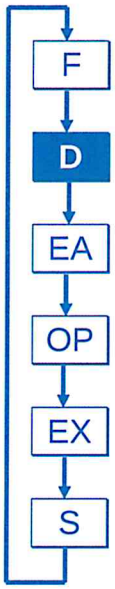
IDENTIFIES OPCODE

4-16 BIT DECODER

DEPENDING ON OPCODE

IDENTIFY OTHER

OPERANDS



## Instruction Processing: EVALUATE ADDRESS

COMPUTE EFFECTIVE ADDRESS

ADD OFFSET TO PC

ADD OFFSET TO BASE REGISTER

## Instruction Processing: FETCH OPERANDS

EXAMPLE:

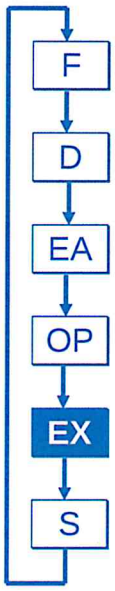
FOR ADD, READ DATA FROM THE REGISTER FILE

## Instruction Processing: EXECUTE

EXAMPLES.

ADD: SEND SIGNAL TO ALU

STORE: (OR LOAD) — DO NOTHING



## Instruction Processing: STORE RESULT

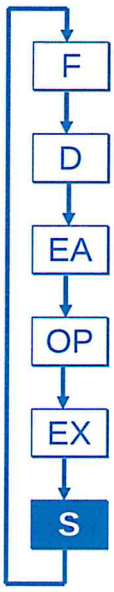
WRITE TO DESTINATION

- REGISTER, OR
- MEMORY

ADDRESS IN MAR

DATA IN MDR

'WRITE' SIGNAL TO MEMORY



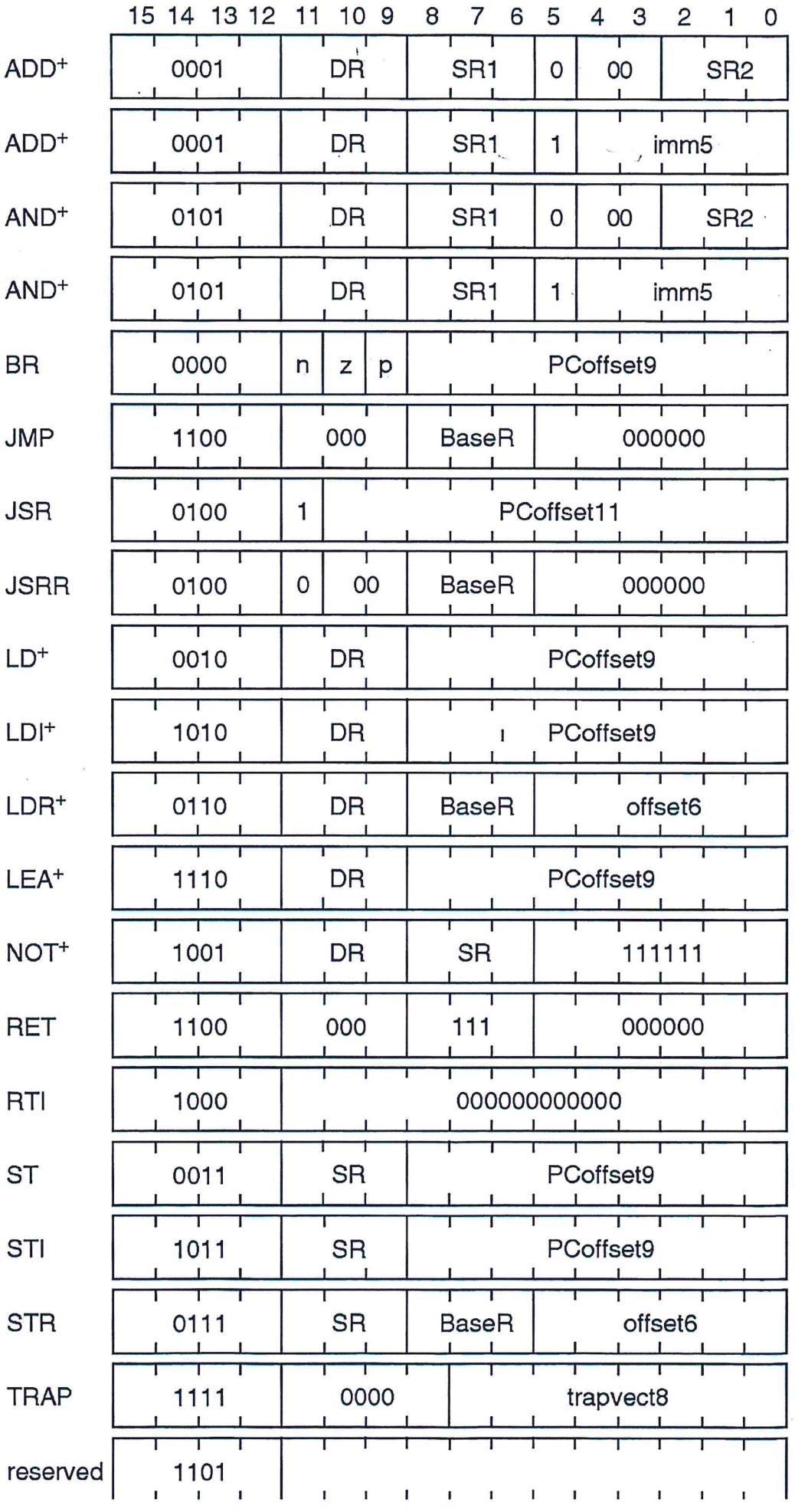
# INSTRUCTIONS (LC3)

LOOK LIKE DATA

3 TYPES OF INSTRUCTIONS:

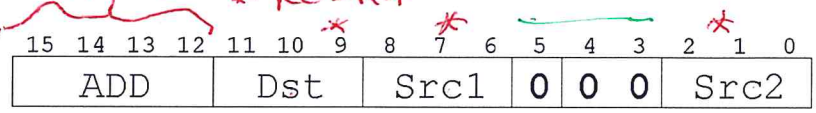
- COMPUTING: ADD, AND
- DATA MOVEMENT: LD, ST
- CONTROL (BRANCH)
  JMP, BR (ex: BRnz)

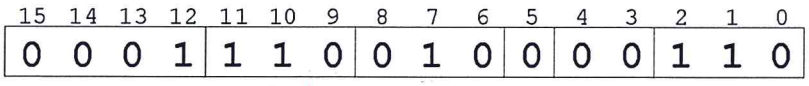
ISA (INSTRUCTION SET ARCHITECTURE)


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
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

## Example: LC-3 Add Instruction

OPCODE * R0-R7




ADD | R6 | R2 | R6

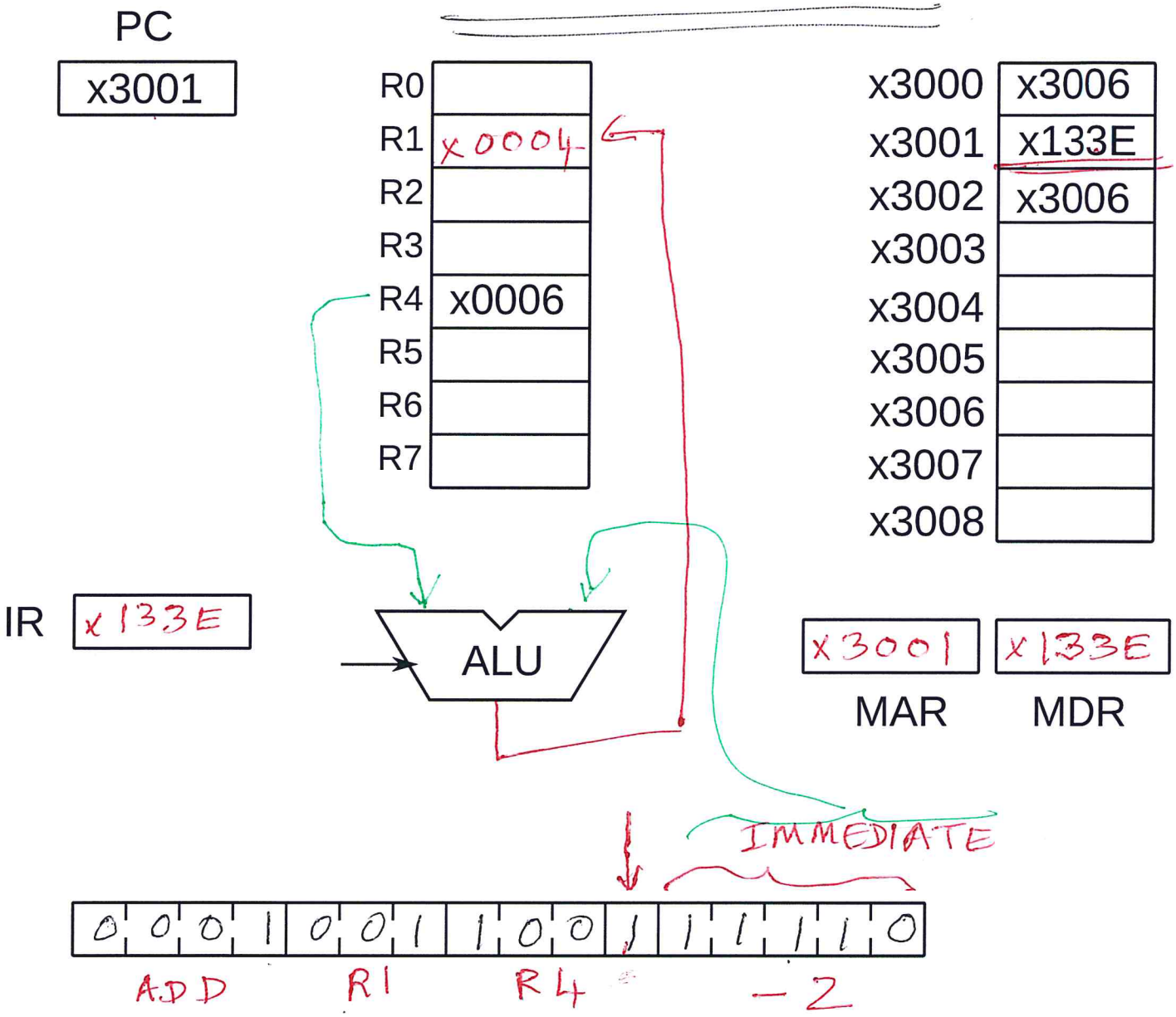
ADD CONTENTS OF R6 AND R2

AND STORE RESULT IN R6

INSTR:

OPCODE: X 1C 8 6

# EXAMPLE ADD




ADD R1, R4, #-2

ADD -2 TO CONTENTS OF R4
AND PUT RESULT IN R1

INSTR: x133E

# CHANGING THE SEQUENCE OF INSTRUCTIONS

IN FETCH, PC IS INCREMENTED

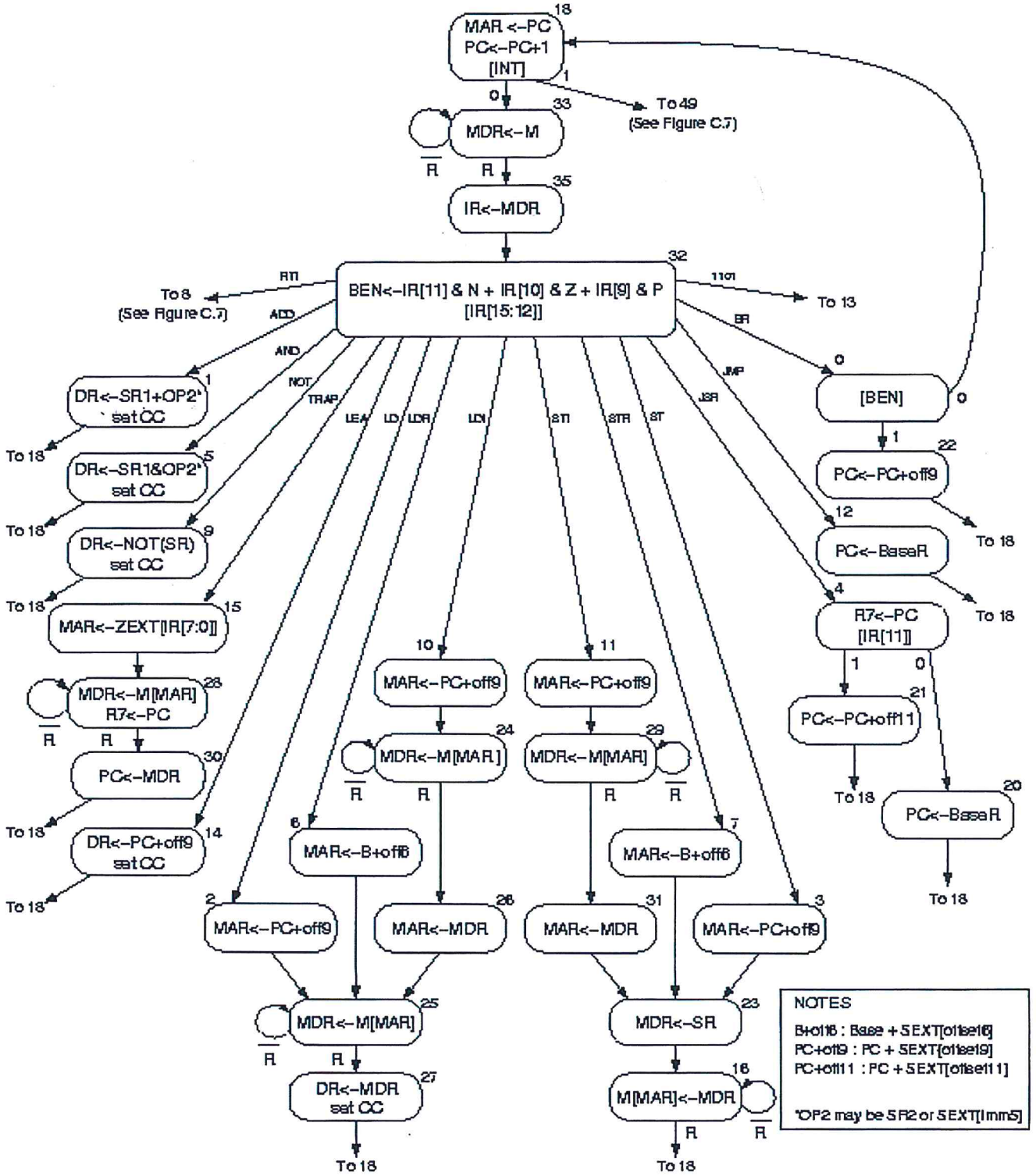
WHAT ABOUT : LOOP

IF, THEN ELSE

JUMPS → ALWAYS CHANGE PC

BRANCH → CONDITIONAL

EXAMPLE: RESULT OF ADD IS 0


18
MAR <-PC
PC<-PC+1
[INT]
1
To 49
(See Figure C.7)
0
33
MDR<-M
R
R
35
IR<-MDR
32
BEN<-IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
RTI
To 8
(See Figure C.7)
ADD
AND
NOT
TRAP
LEA
LD
LDR
LDI
STI
STR
ST
JMP
JSR
BR
1101
To 13
0
[BEN]
0
1
22
PC<-PC+off9
To 18
12
PC<-BaseR
To 18
4
R7<-PC
[IR[11]]
1
0
21
PC<-PC+off11
To 18
20
PC<-BaseR
To 18
1
DR<-SR1+OP2*
set CC
To 18
5
DR<-SR1&OP2*
set CC
To 18
9
DR<-NOT(SR)
set CC
To 18
15
MAR<-ZEXT[IR[7:0]]
28
MDR<-M[MAR]
R7<-PC
R
R
30
PC<-MDR
To 18
14
DR<-PC+off9
set CC
To 18
10
MAR<-PC+off9
24
MDR<-M[MAR]
R
R
6
MAR<-B+off6
2
MAR<-PC+off9
26
MAR<-MDR
11
MAR<-PC+off9
29
MDR<-M[MAR]
R
R
7
MAR<-B+off6
31
MAR<-MDR
3
MAR<-PC+off9
25
MDR<-M[MAR]
R
R
27
DR<-MDR
set CC
To 18
23
MDR<-SR
16
M[MAR]<-MDR
R
R
To 18
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
PC+off11 : PC + SEXT[offset11]
*OP2 may be SR2 or SEXT[Imm5]

# 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*

# ADD/AND (Register)

*this zero means "register mode"*

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD | 0 | 0 | 0 | 1 | Dst | | | Src1 | | | 0 | 0 | 0 | Src2 | | |
| AND | 0 | 1 | 0 | 1 | Dst | | | Src1 | | | 0 | 0 | 0 | Src2 | | |