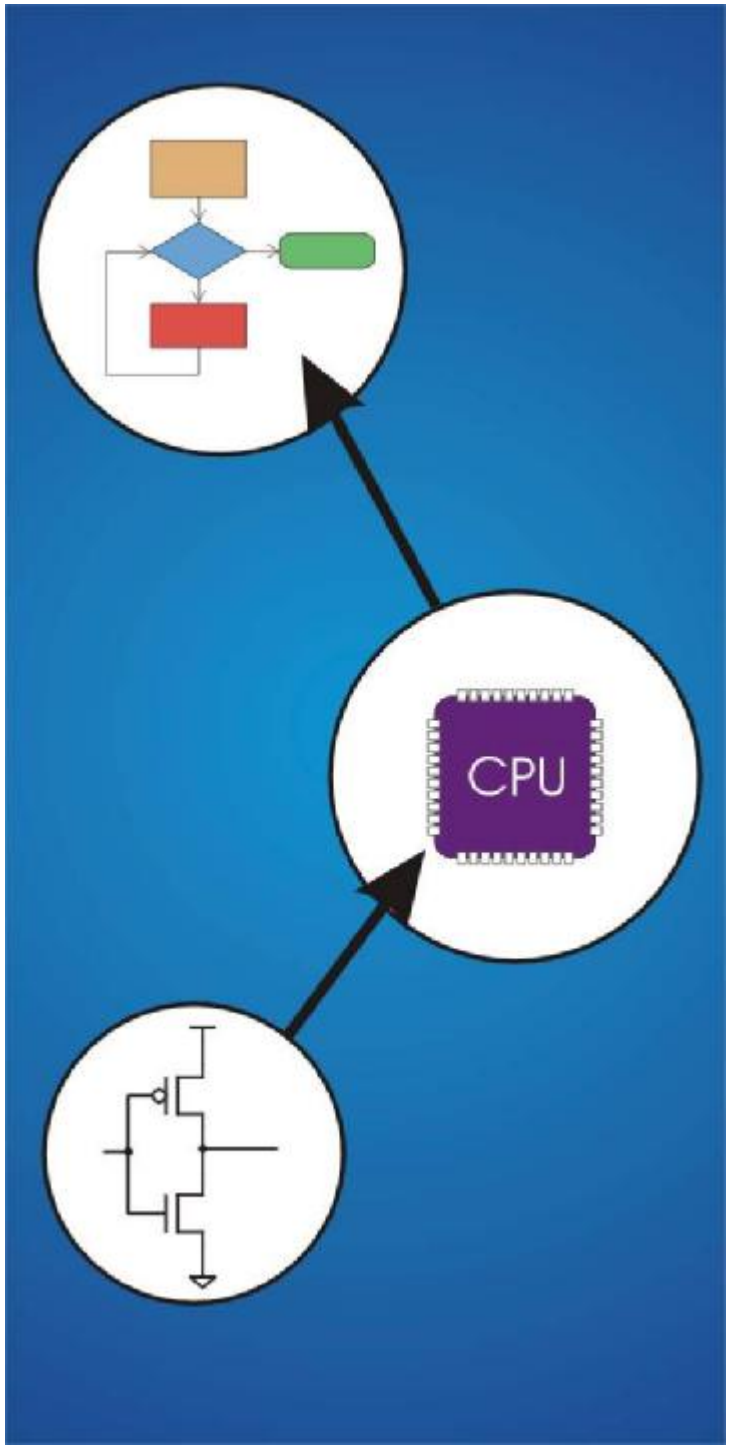


第五章
LC-3结构

计算机系统的抽象层次

Problem Specification

compute the fibonacci sequence

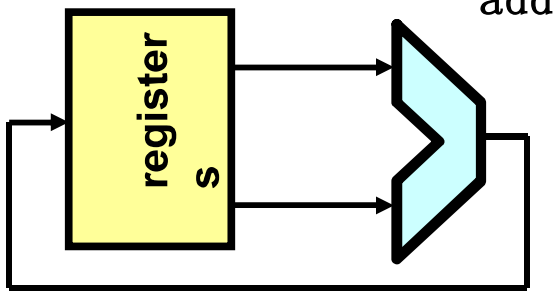
Algorithm Program

```
for(i=2; i<100; i++) {
  a[i] = a[i-1]+a[i-2];}
```

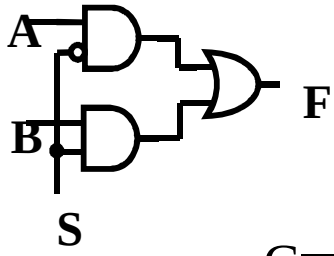
ISA (Instruction Set Architecture)

```
load r1, a[i];
add  r2, r2, r1;
```

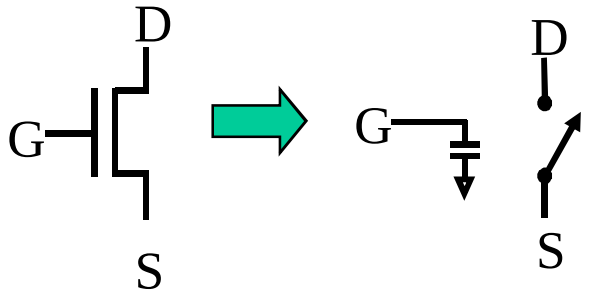
microArchitecture



Logic



Transistors



Physics/Chemistry

抽象层次

本章通过一个简单的计算机系统实例LC-3给大家介绍更高的抽象层次ISA（指令集结构）.

ISA为机器语言程序员提供了有关控制机器所需要所有必要信息。或给高级语言编译器开发者提供将高级语言转换成机器代码的必要信息。

ISA是计算机硬件和软件的分界面.

Instruction Set Architecture(ISA):指令集结构

ISA = 向以机器语言编程的程序员提供有关控制机器所需要的所有必要信息 。包括内存组织方式、寄存器组、指令集等信息。

- 内存组织方式
  - Ø 寻址空间 – 有多少个存储空间?
  - Ø 寻址能力 – 每个存储空间有多少位?
- 寄存器组
  - Ø 有多少? 存储数据长度? 怎么使用?
- 指令集
  - Ø 操作码
  - Ø 数据类型
  - Ø 寻址模式

LC-3 Overview: 内存组织和寄存器

内存组织

- 寻址空间: 2^{16} 个存储单元 (16位地址)
- 寻址能力: **16 bits**

寄存器组

- 提供一个快速的临时存储空间, 通常可在一个机器周期的时间访问到
  - Ø访问存储器的时间往往远大于一个机器周期的时间
- 8个通用寄存器: **R0 - R7**
  - Ø每个可存储数据宽度为**16 bits**
  - Ø寄存器编址需要多少位二进制？
- 其它寄存器
  - Ø程序员不能直接访问，被指令使用或影响
  - ØPC (program counter), condition codes，MDR，MAR

LC-3 Overview: 指令集

操作码

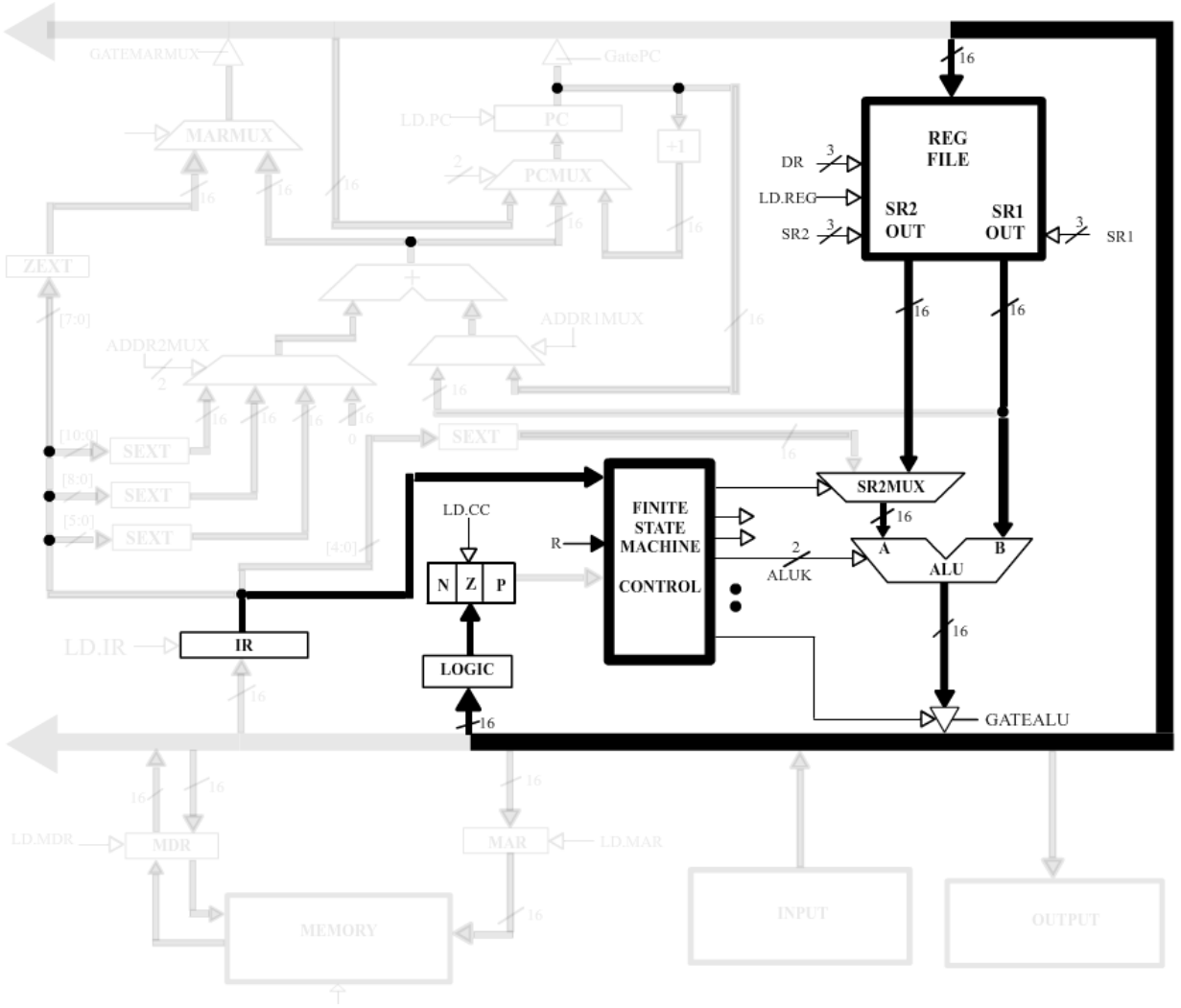
- **15 个操作码(P79)**
- 逻辑和运算指令: **ADD(0001), AND(0101), NOT(1001)**（助记符）
- 数据搬移指令: **LD(0010), LDI(1010), LDR(0110), LEA(1110), ST(0011), STR(0111), STI(1011)**
- 控制指令: **BR(0000), JSR/JSRR(0100), JMP(1100), RTI(1000), TRAP(1111)**
- 目的操作数为寄存器的指令会根据写入寄存器的值设置条件码
  - Ø**N =** 写入值为负**(<0), Z =**写入值为零**(=0), P =**写入值为正**(> 0)**
  - Ø组合 **NZ(<=0)/ZP(>=0)/NP(<>0)/NZP(?)**

数据类型： **16**位定点补码整数

寻址方式：指令中指示参与运算操作数存储位置的方法

- 非内存寻址（操作数不在内存中）：
  直接寻址（操作数在指令中）, 寄存器寻址（操作数在寄存器中）
- 内存寻址（操作数在内存中）：***PC-相对***, 间接, 基址***+偏移***

条件码(NZP 逻辑：只有写寄存器的指令才影响NZP标志)



运算指令

LC-3只支持三个运算指令: ADD, AND, NOT

实现特点：

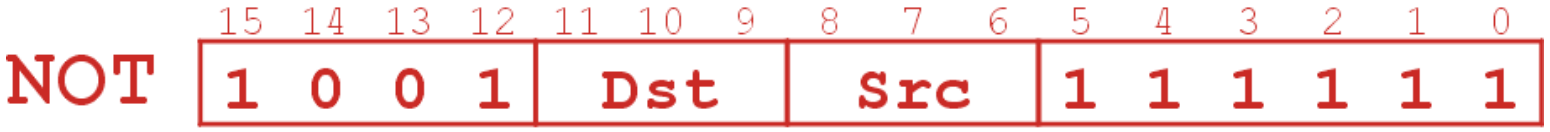
源操作数和目的操作数都是寄存器

- **这些指令的操作数不能够直接使用在内存的数据.**
- **ADD and AND 可以支持“立即数”模式,**
  一个源操作数可以直接在指令中给出.
- **内存中的数参与运算需要利用数据搬移指令实现搬移到寄存器中，运算结果也需要利用数据搬移指令搬移回内存中**

Will show dataflow diagram with each instruction.

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

NOT (SRC/DST两个操作数必须是寄存器)



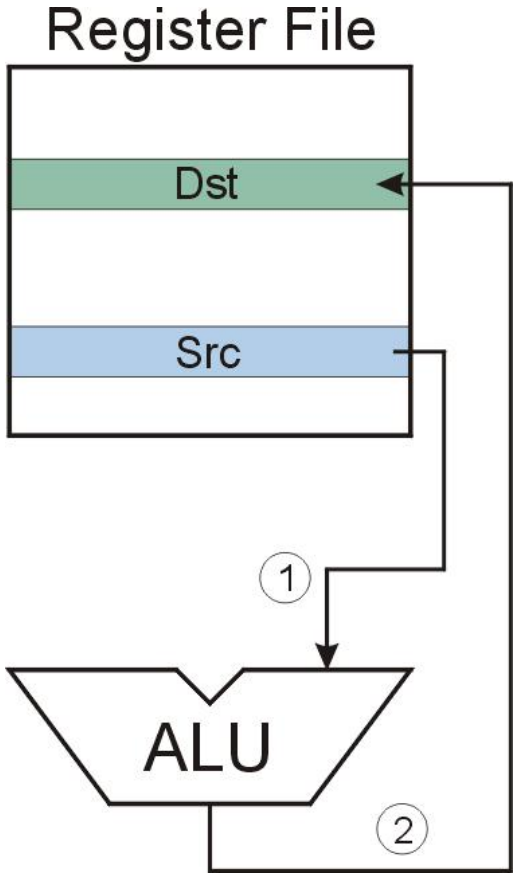
Note: Src 和 Dst
可以是同一个寄存器

NOT R2,R3

1001 010 011 111111

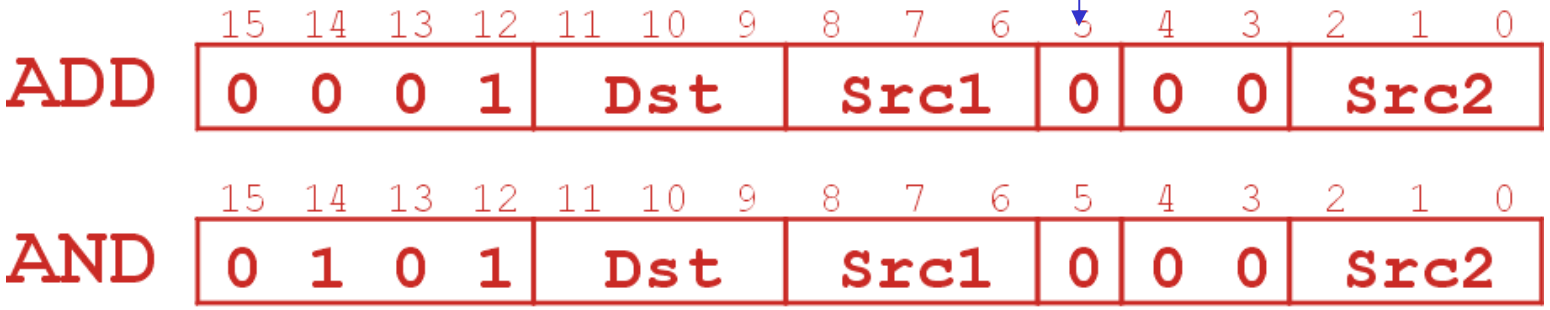
NOT R2,R2

1001 010 010 111111



ADD/AND (寄存器模式)

为0指示"register mode"



Note: 寄存器模式，两个源操作数和1个目的操作数都为寄存器。

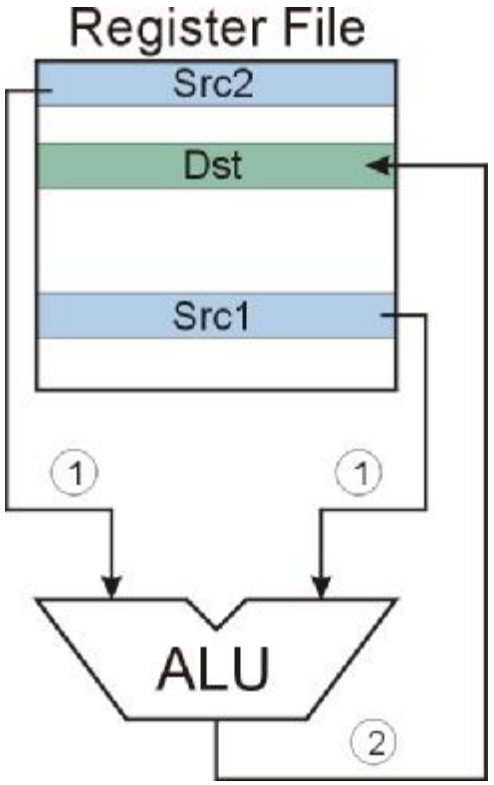
Note: Src1/2 和 Dst 可以是同一个寄存器

ADD R1,R2,R3

0001 001 010 000 011

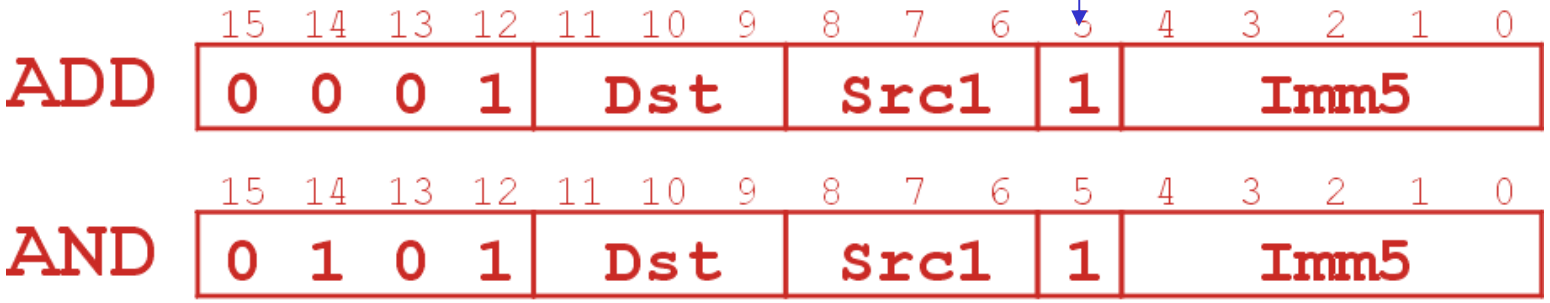
ADD R1,R1,R3

ADD R1,R1,R1



ADD/AND (立即数模式)

为1指示 “immediate mode”

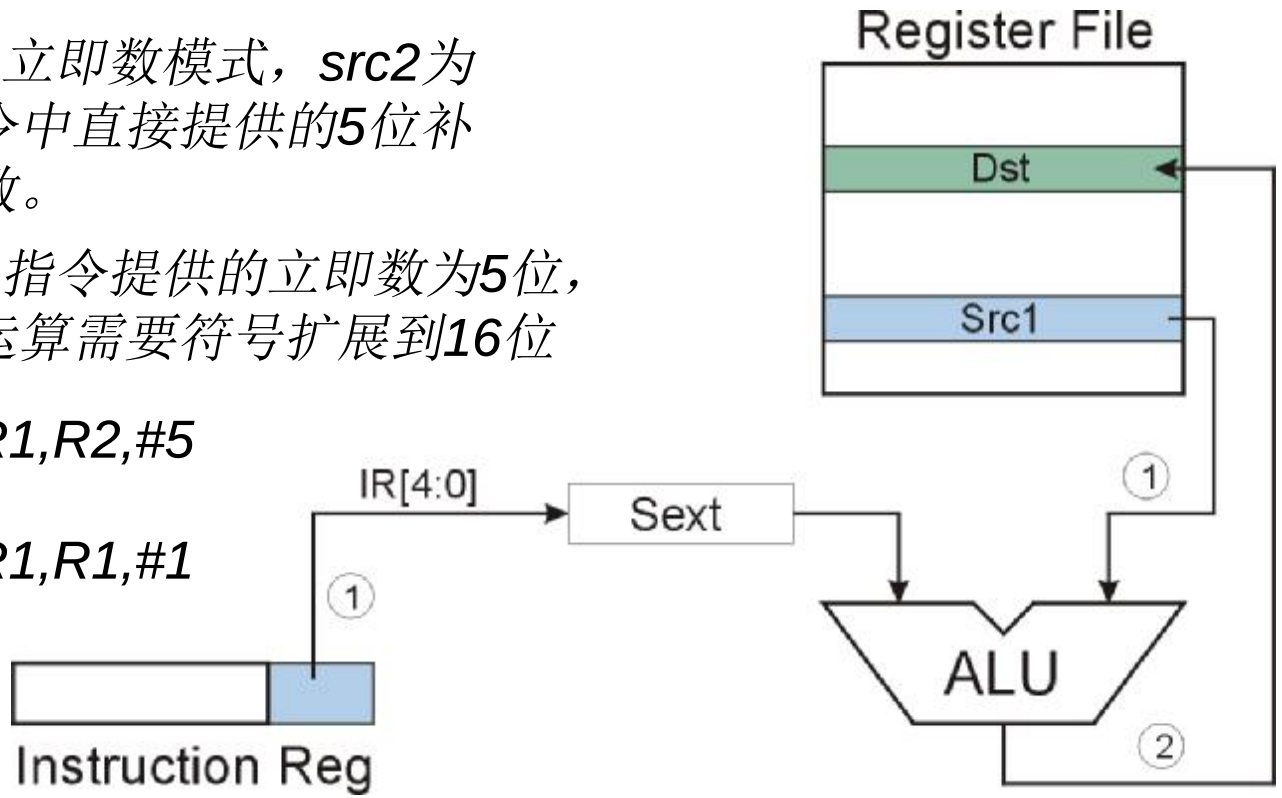


Note: 立即数模式，src2为在指令中直接提供的5位补码整数。

Note: 指令提供的立即数为5位，参与运算需要符号扩展到16位

ADD R1,R2,#5

AND R1,R1,#1



运算指令的使用

只使用**ADD, AND, NOT…**

- 怎么做减法?
- 怎么实现 OR操作?
- 怎么把一个寄存器的值赋给另外一个?
- 怎么初始化一个寄存器的值为0?

数据搬移指令

Load – 从内存中读数据到寄存器中

按内存数的寻址方式不同可分为：

- **LD:** PC-相对寻址模式
- **LDR:** 寄存器基址+偏移模式
- **LDI:** 间接寻址模式

Store – 写寄存器值到内存

按内存数的寻址方式不同可分为：

- **ST:** PC-相对寻址模式
- **STR:**寄存器基址+偏移模式
- **STI:** 间接寻址模式

LEA - 计算操作数的有效地址，存放到寄存器

- **LEA:** 用立即数的方式给出操作数相对PC的偏移
- ***LEA指令不访存***

PC相对寻址模式

LC-3指令长度16位，内存地址长度16位，能在指令中直接给出内存数的有效地址吗？

- **16位指令中操作码占用4 bits，一个目的寄存器需要占用 3 bits ，只剩下9位来编码地址了**
- **不可能直接给出16位地址! 怎么解决**

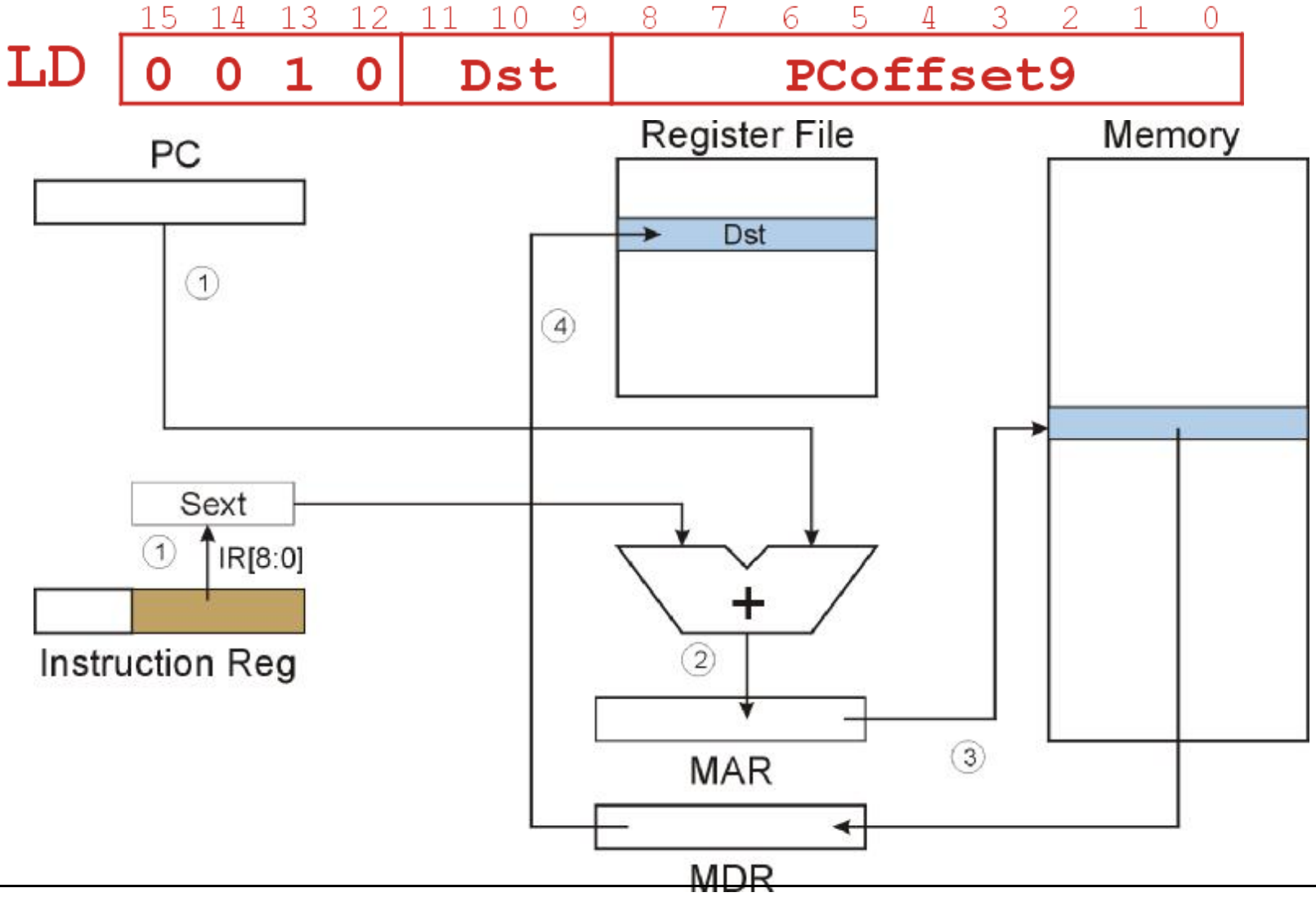
解决方法:

- **利用PC寄存器**
- **剩下的 9 bits 用来表示数据地址和 PC的偏移量(offset).**
- **数据有效地址为PC+offset**
- **局限性：9 bits:** $-256 \le \text{offset} \le +255$

数据的地址范围为： $PC - 256 \le X \le PC + 255$

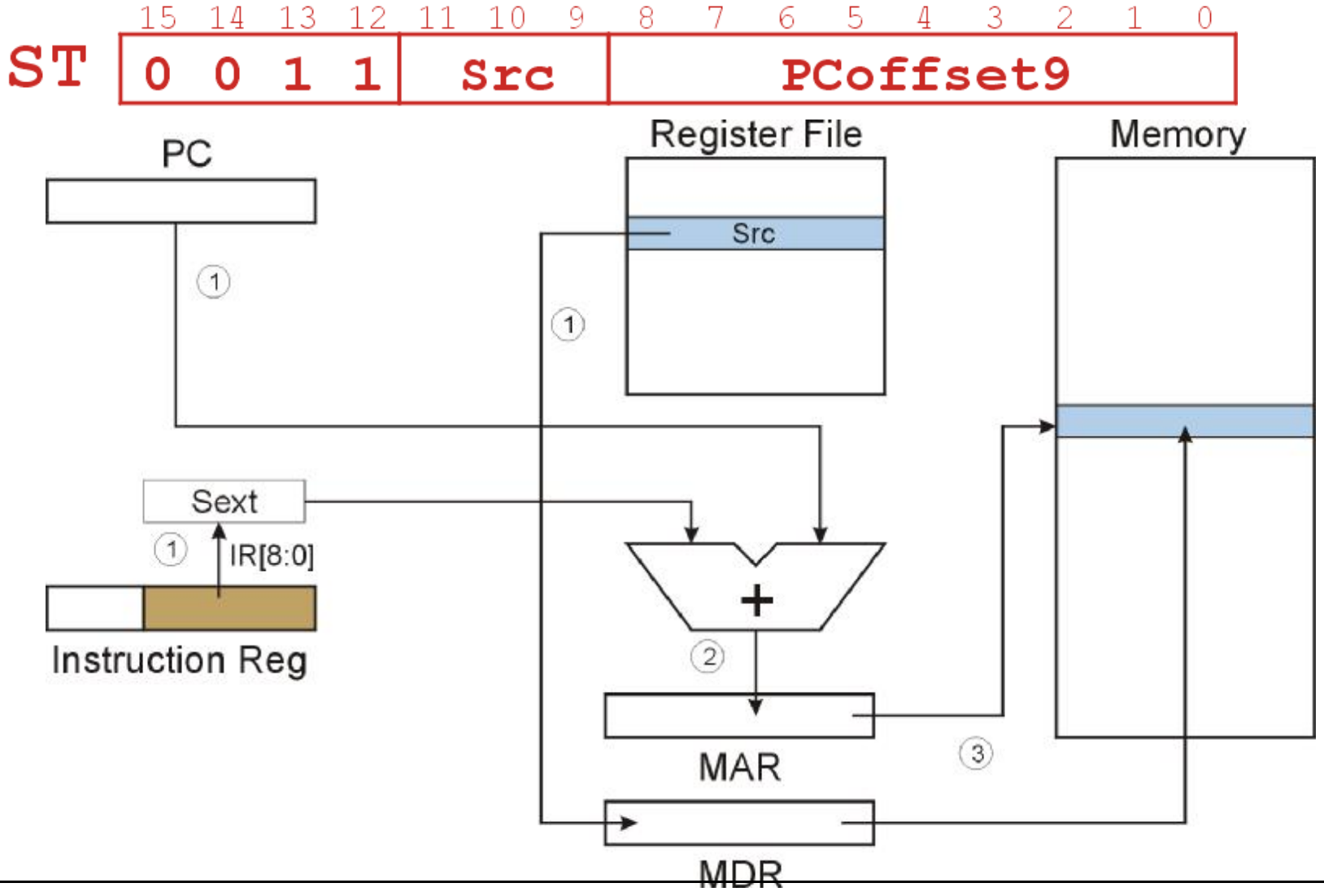
注意：PC不是当前指令的地址，而是下一条指令的地址

LD (PC-Relative)



x30F6 **0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 1** *x3000指令访问x3100?*

ST (PC-Relative)



x30F6 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1

（基于PC的）间接寻址模式

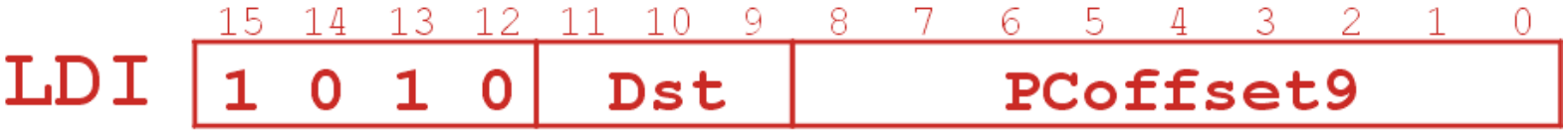
LC-3的PC相对寻址模式, 只能访问PC前或后256的内存单元

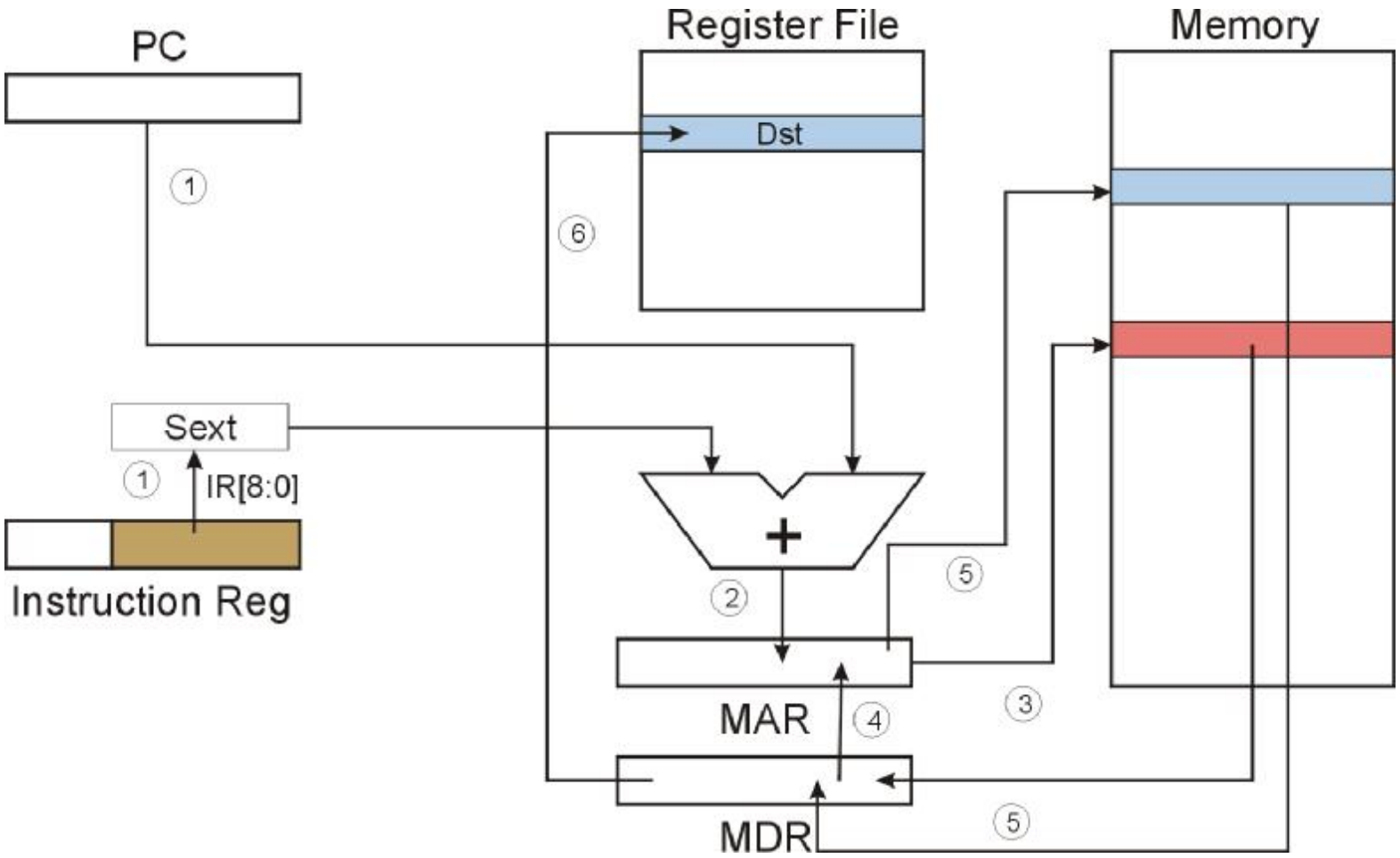
- 剩下的内存怎么访问?

解决方案 #1:

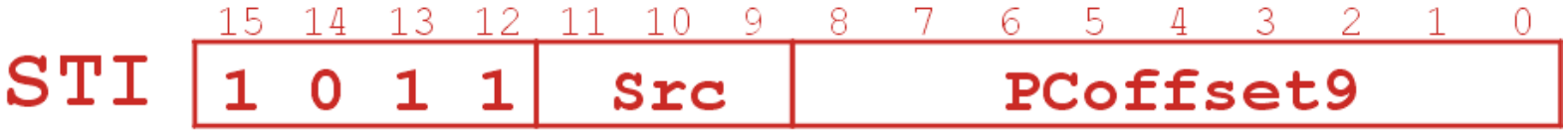
- 在PC相对寻址能访问到的内存单元存放一个16位地址（不是数据了）.先读取这个地址，然后以这个地址去访问内存。
- 类似C语言的指针

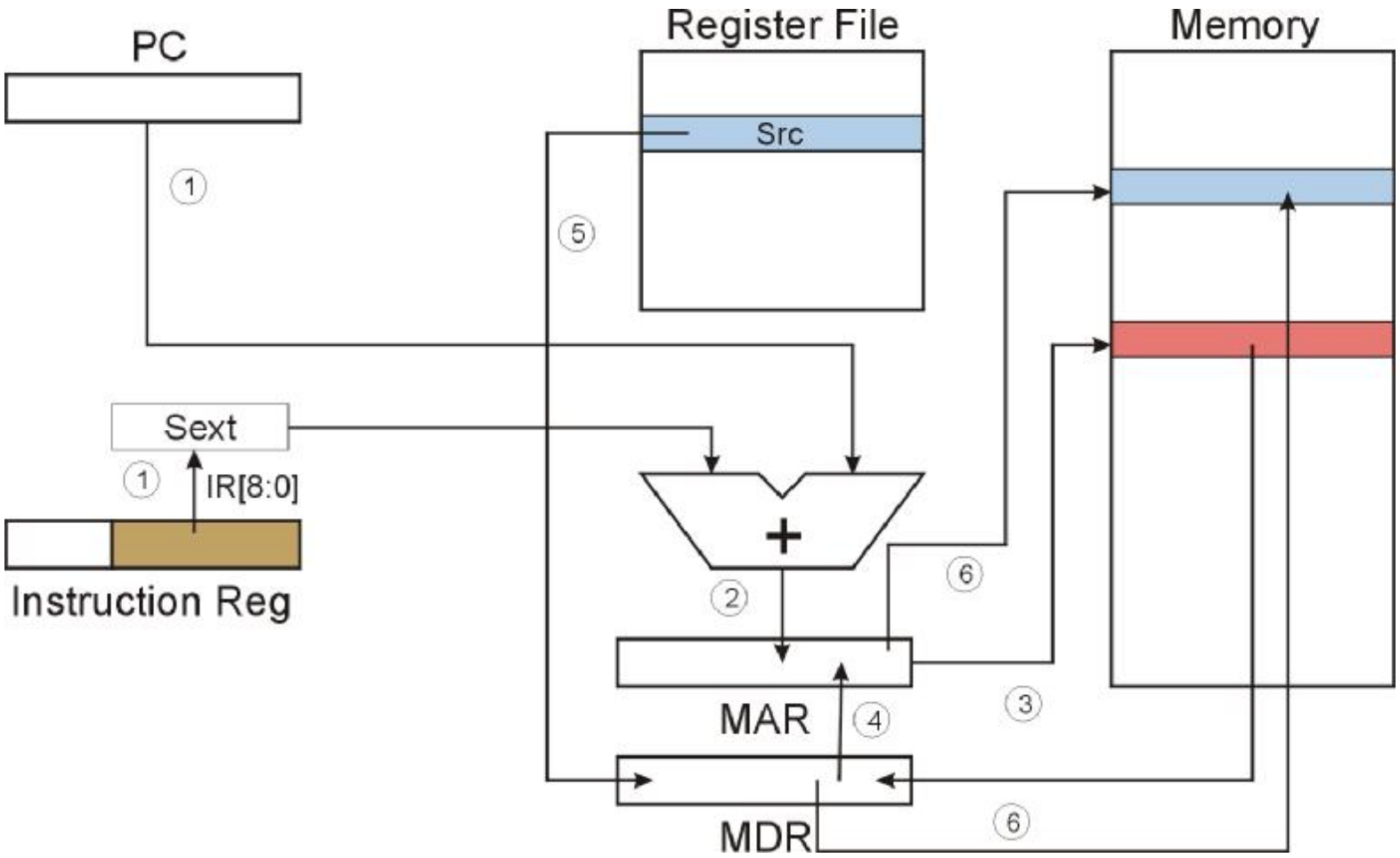
LDI (间接寻址)


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
LDI
1 0 1 0
Dst
PCoffset9


PC
Register File
Memory
Dst
1
6
Sext
1
IR[8:0]
Instruction Reg
+
2
5
MAR
4
3
MDR
5

STI (间接寻址)


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
STI
1 0 1 1
Src
PCoffset9


PC
Register File
Memory
Src
1
5
Sext
1
IR[8:0]
Instruction Reg
+
2
6
MAR
4
3
MDR
6

（寄存器）基址偏移寻址模式

LC-3的PC相对寻址模式, 只能访问PC前或后256的内存单元

- 剩下的内存怎么访问?

解决方法 #2:

- 利用寄存器存放一个16位地址.偏移以立即数形式存放在指令中
- 类似C语言的数组。**int a[10]; *(a+2)**

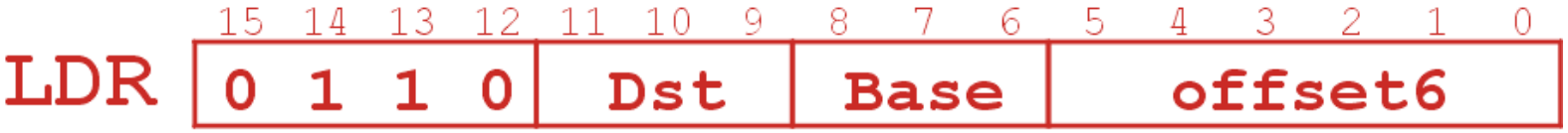
指令编码:

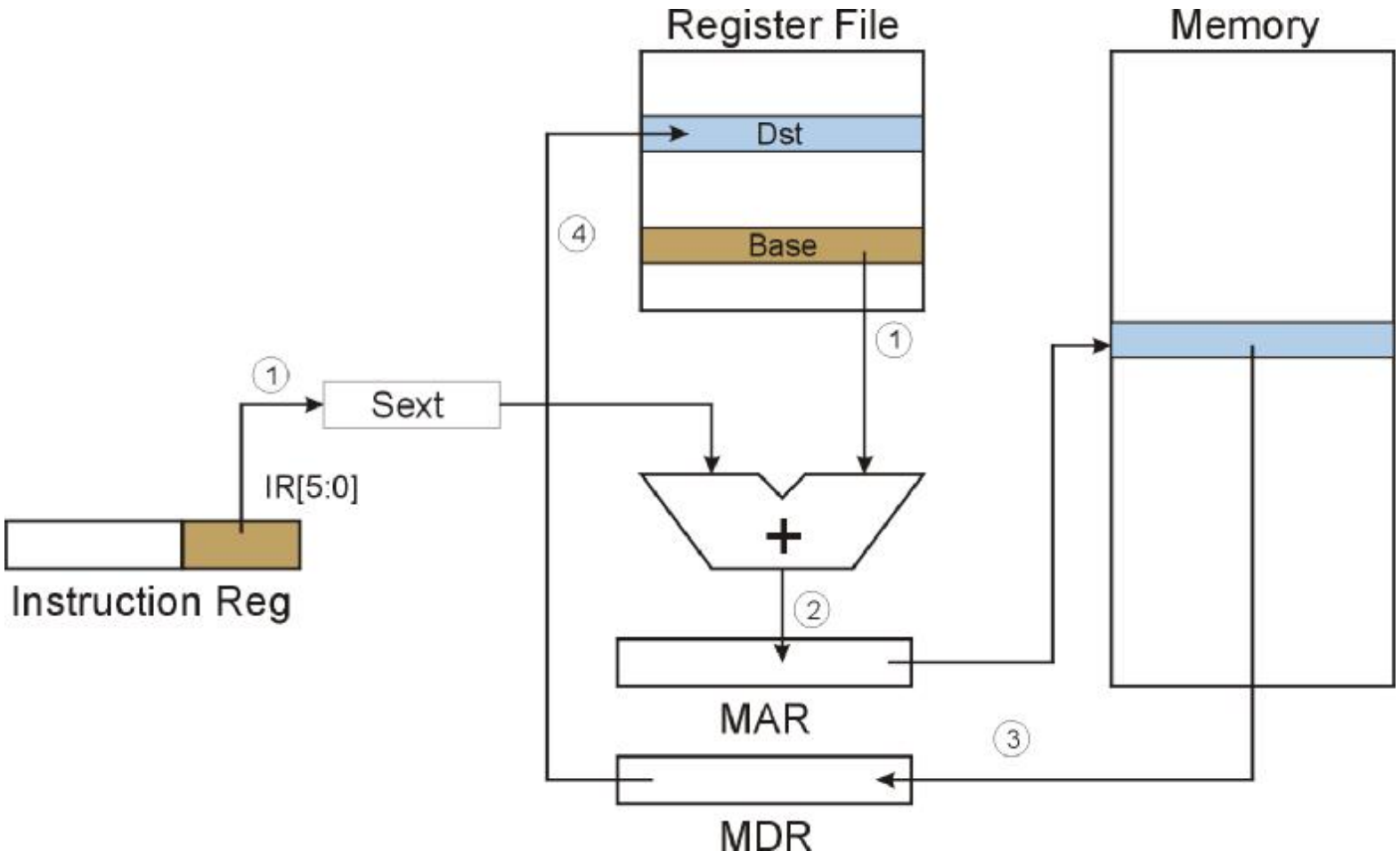
操作码占用4 bits, 寄存器占用3bits ， 基址寄存器占用3 bits

剩下6bits可用作偏移。$-32 \leq \text{offset} \leq +31$

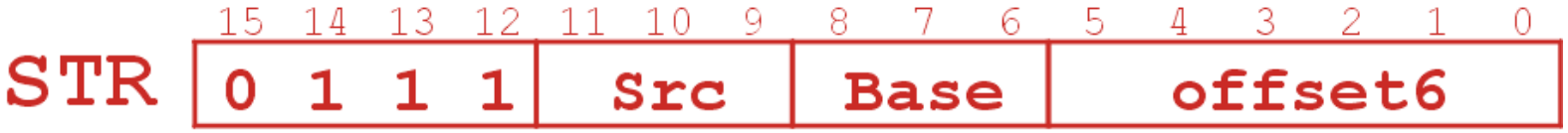
基址寄存器中如何设置16位地址？ LD、LEA

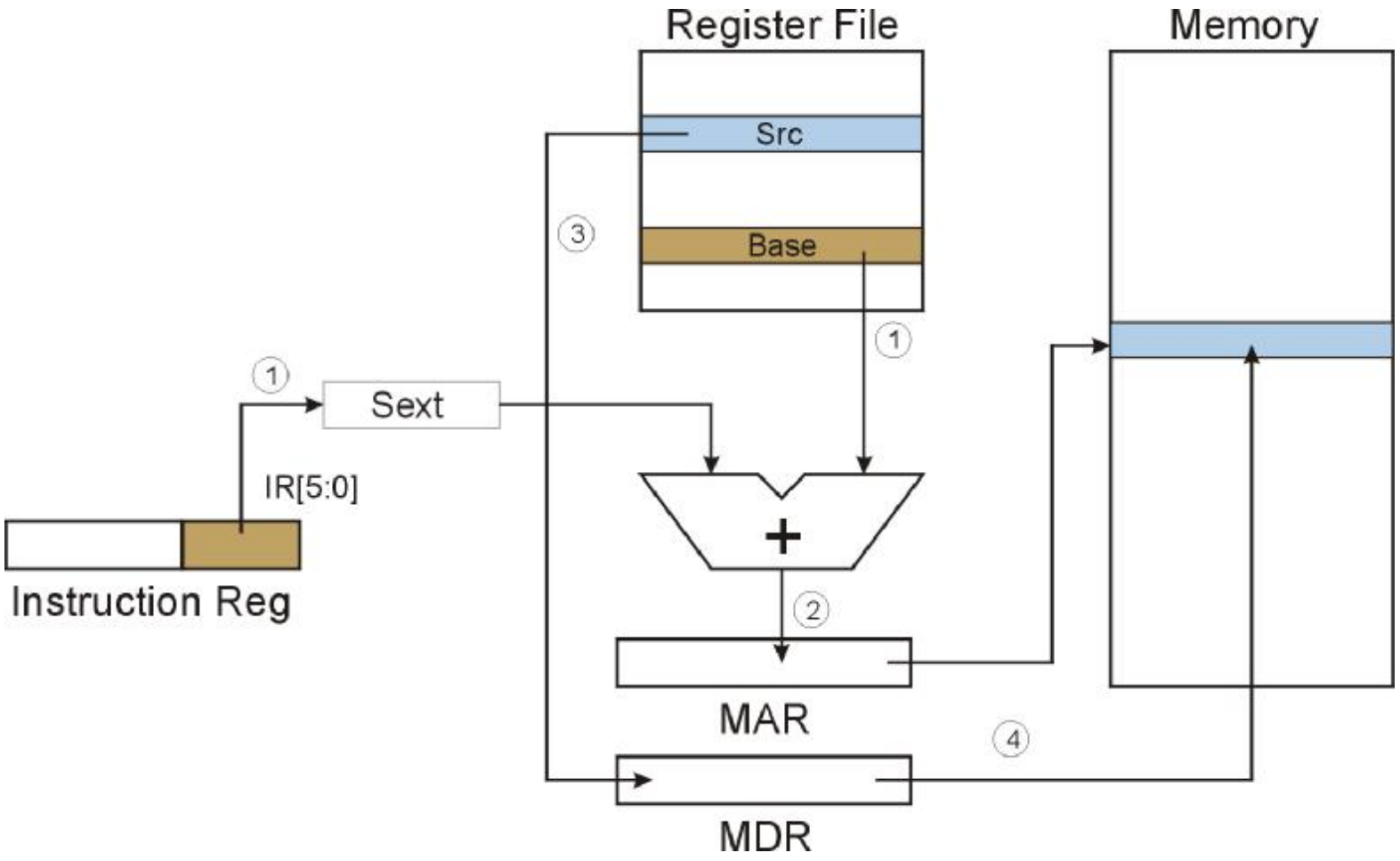
LDR (基址偏移寻址模式)





STR (基址偏移寻址模式)





LEA :Load Effective Address 计算有效地址

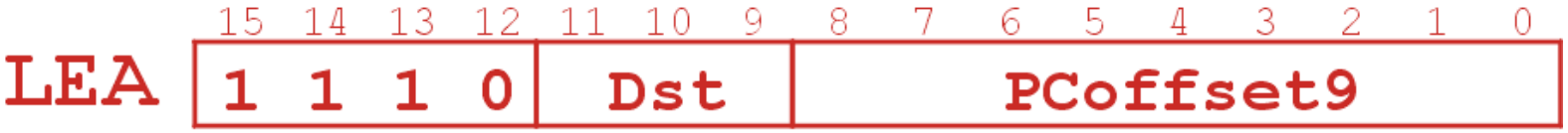
计算操作数的有效地址，存放到寄存器

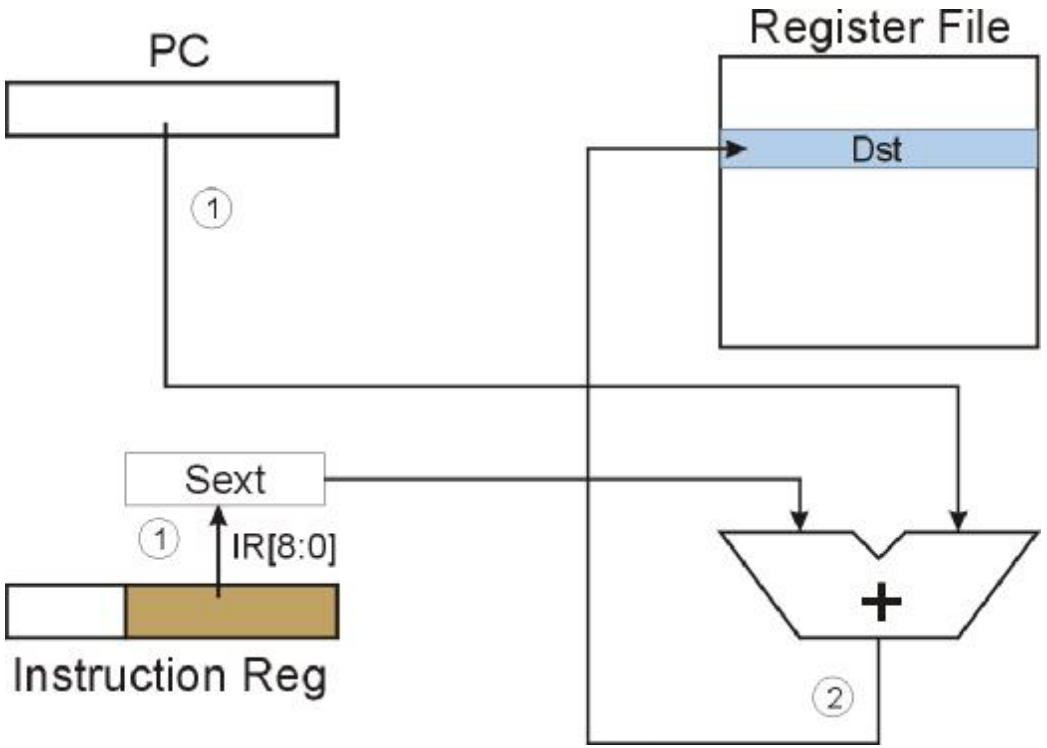
- **LEA:** 用立即数的方式给出操作数相对**PC**的偏移**(9bits)**
- 计算方法：**PC+offset ->**寄存器
- ***LEA指令不访存***

Note: 寄存器里面存放的是地址,而不是内存单元存放的数据。

应用:访问连续的数据区域,用**LEA** 指令得到数据区域的起始地址,然后用**LDR**指令访问.

LEA (Immediate)





Example

	Address	opcode	Instruction	Comments
LEA	x30F6	1 1 1 0	0 0 1 1 1 1 1 1 1 1 0 1	*R1 ¬ PC – 3 = x30F4*
ADD	x30F7	0 0 0 1	0 1 0 0 0 1 1 0 1 1 1 0	*R2 ¬ R1 + 14 = x3102*
ST	x30F8	0 0 1 1	0 1 0 1 1 1 1 1 1 0 1 1	*M[PC - 5] ¬ R2* *M[x30F4] ¬ x3102*
AND	x30F9	0 1 0 1	0 1 0 0 1 0 1 0 0 0 0 0	*R2 ¬ 0*
ADD	x30FA	0 0 0 1	0 1 0 0 1 0 1 0 0 1 0 1	*R2 ¬ R2 + 5 = 5*
STR	x30FB	0 1 1 1	0 1 0 0 0 1 0 0 1 1 1 0	*M[R1+14] ¬ R2* *M[x3102] ¬ 5*
LDI	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*

控制指令

通过更新PC，改变程序执行顺序

条件跳转

- 跳转到分支仅当指定的条件成立
  - Ø更新PC到分支地址，通过在当前的PC值加一个偏移实现
- 否则, 不跳转到分支
  - ØPC 不改变，顺序执行下一条指令

无条件跳转(直接跳转)

- PC值肯定被改变到目标地址

TRAP（陷入指令）

- 改变PC 到操作系统提供的服务子程序的入口地址。“service routine”
- 服务子程序完成后返回到TRAP指令后一条程序代码继续执行。

条件码

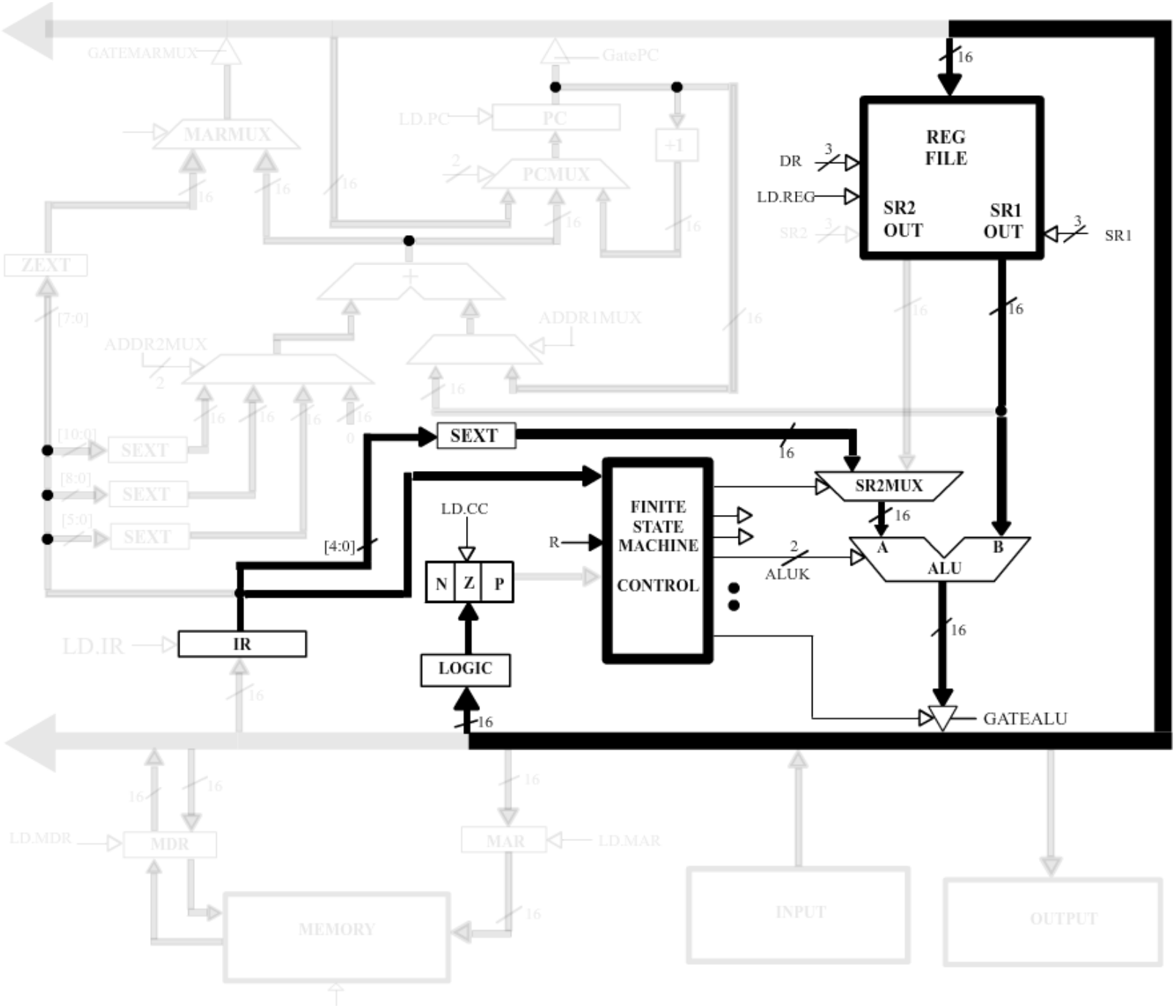
LC-3 有3个1位的条件码寄存器，由最近写入的寄存器值确定

N – negative（〈0）
Z – zero　　（=0）
P -- positive　(〉0)

N Z P 同一时刻只有一个标志位会改变

- 由最近写入的寄存器值确定
- 任何一条写寄存器的指令都会改变条件码 **(ADD, AND, NOT, LD, LDR, LDI, LEA)**
- **Store指令和控制指令不改变条件码**

```
AND R1,R1,#0      NOT   R1,R1     NOT  R1,R1
 x3000:LEA R1,PC+25
ADD  R1,R1,#-1
ADD  R1,R1,#15
```


GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
2
PCMUX
16
REG
FILE
DR
3
LD.REG
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
LD.CC
N
Z
P
R
FINITE
STATE
MACHINE
CONTROL
SR2MUX
A
B
ALU
ALUK
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

条件跳转指令

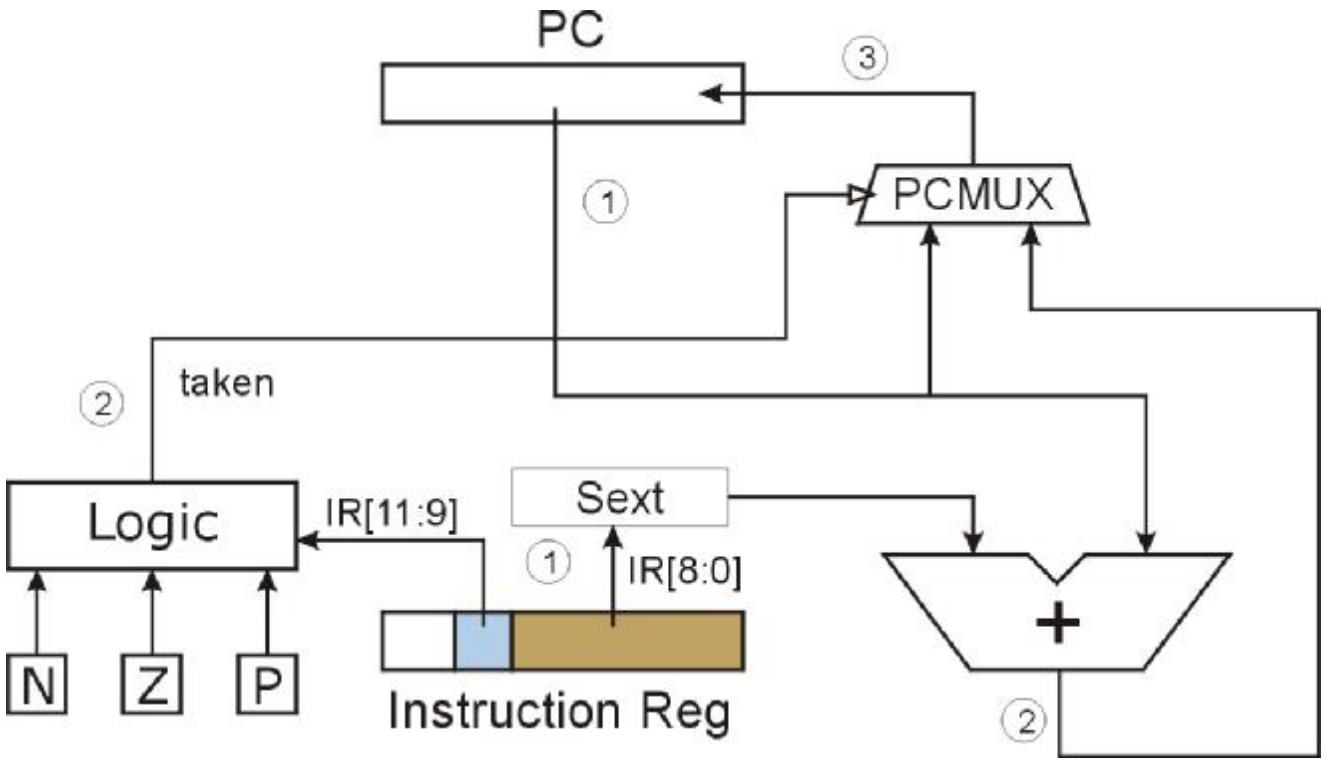
在跳转指令中指示需要检测哪个条件码**(IR[11:9])**（可以是一个或者多个），如果指定的条件码成立，则跳转，否则不跳转。

- 目标地址采用了**PC**相对寻址
  target address = PC + offset (IR[8:0])
- **Note: PC**不是当前指令地址，而是下一条指令的地址。
- **Note:** 只能跳转到跳转指令的前**255**条指令或后**256**条指令。
- **Note:** 必须和上一条会修改寄存器的指令配合使用

Ex:

- **X3100: 0001 001 001 1 11111**
- **X3101 : 0000 010 0000 00100**

BR (PC相对寻址)



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

指令助记: BRn /BRz /BRp /BRnzp /BRnz /BRnp /BRzp

应用-discussion

1　判断 r0的值是否为5, 等于5则跳转。

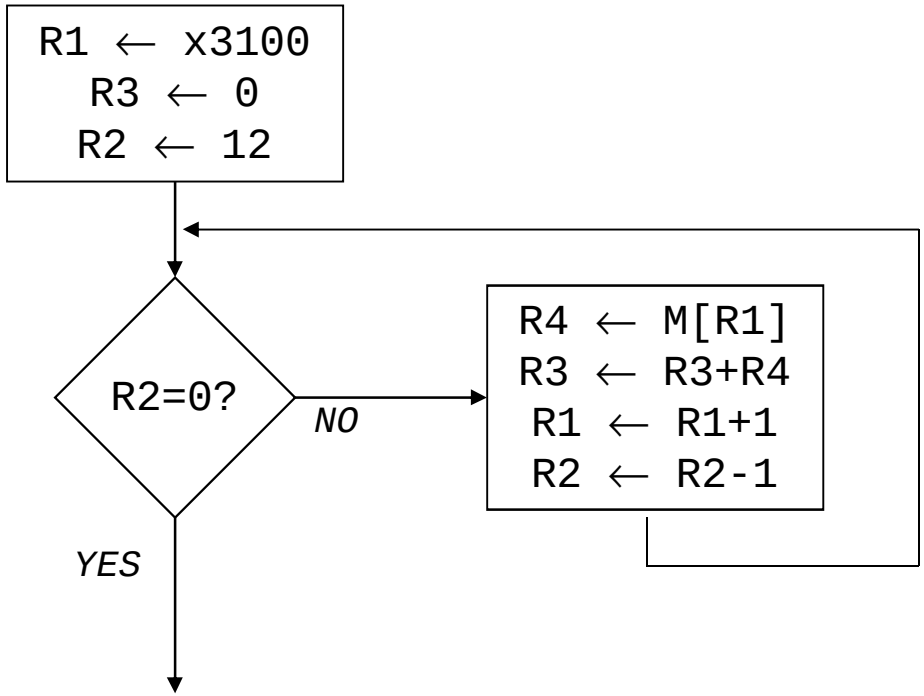
2　判断R0的值>5,>=5,<5 ,<=跳转

3　判断R0的值>40跳转

4　比较R0,R1是否相等跳转

跳转指令的应用：循环控制

计算12个整数的和：
整数存放的起始地址：x3100. 程序起始地址： x3000。



Sample Program

Address	*Instruction*	*Comments*
x3000	1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1	*R1 ¬ x3100 (PC+0xFF)*
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	*R3 ¬ 0*
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ¬ 0*
x3003	0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0	*R2 ¬ 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 1 0 1 0	*Goto x3004 (PC-6)*

哨兵法：事前不确定循环次数

计算若干个正整数的和：

整数存放的起始地址：**x3100**，以一个负数结尾。

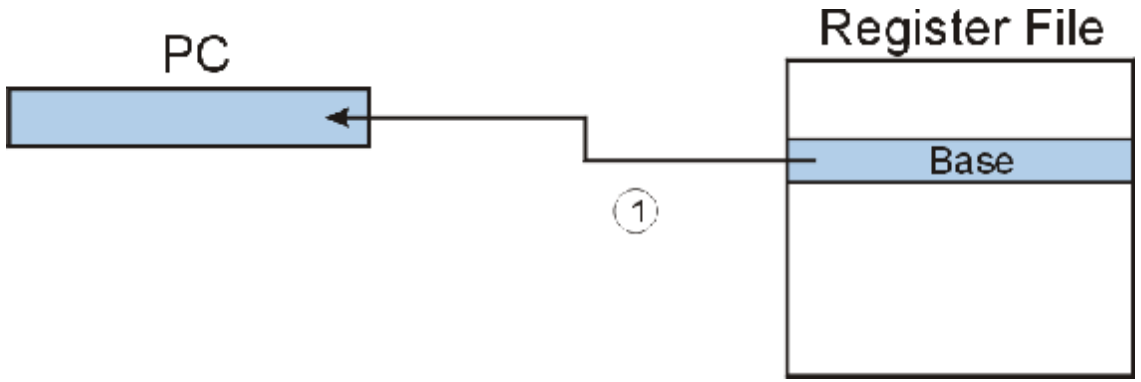
程序起始地址： **x3000**。**(R3:**和，**R1:**数据起始地址，**R4:**数据**)**

Address	*Instruction*	*Comments*
x3000	1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1	*R1 ¬ x3100 (PC+0xFF)*
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	*R3 ¬ 0*
x3002	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	*R4 ¬ M[R1]*
x3003	0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0	*BRn 3008;*
x3004	0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0	*R3 ¬ R3+R4*
x3005	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*R1 ¬ R1+1*
x3006	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	*R4 ¬ M[R1]*
x3007	0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1	*BRnzp x3003*
X3008	halt	
x3009		

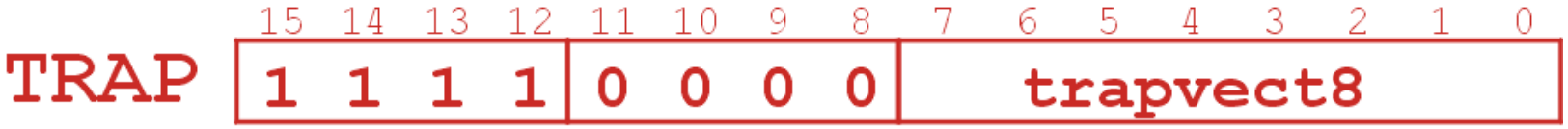
JMP (寄存器存放跳转地址)

JMP是一个绝对跳转指令 – 总是跳转

- 跳转的目标地址存放在寄存器中
- 寄存器可以存放16位地址。可以跳转到任何地方。
- 条件跳转是有局限性的：**-256到+255**



TRAP：调用系统服务程序



调用系统服务程序,服务程序由 **8-bit “trap vector”**指定，总共支持**256**个服务程序。

vector	*routine*
x23	**input a character from the keyboard to R0**
x21	**output a character in R0 to the monitor**
x25	**halt the program**

调用结束后, **PC**被设置成当前**TRAP** 指令的下一条。

(具体原理后面会讨论**)**

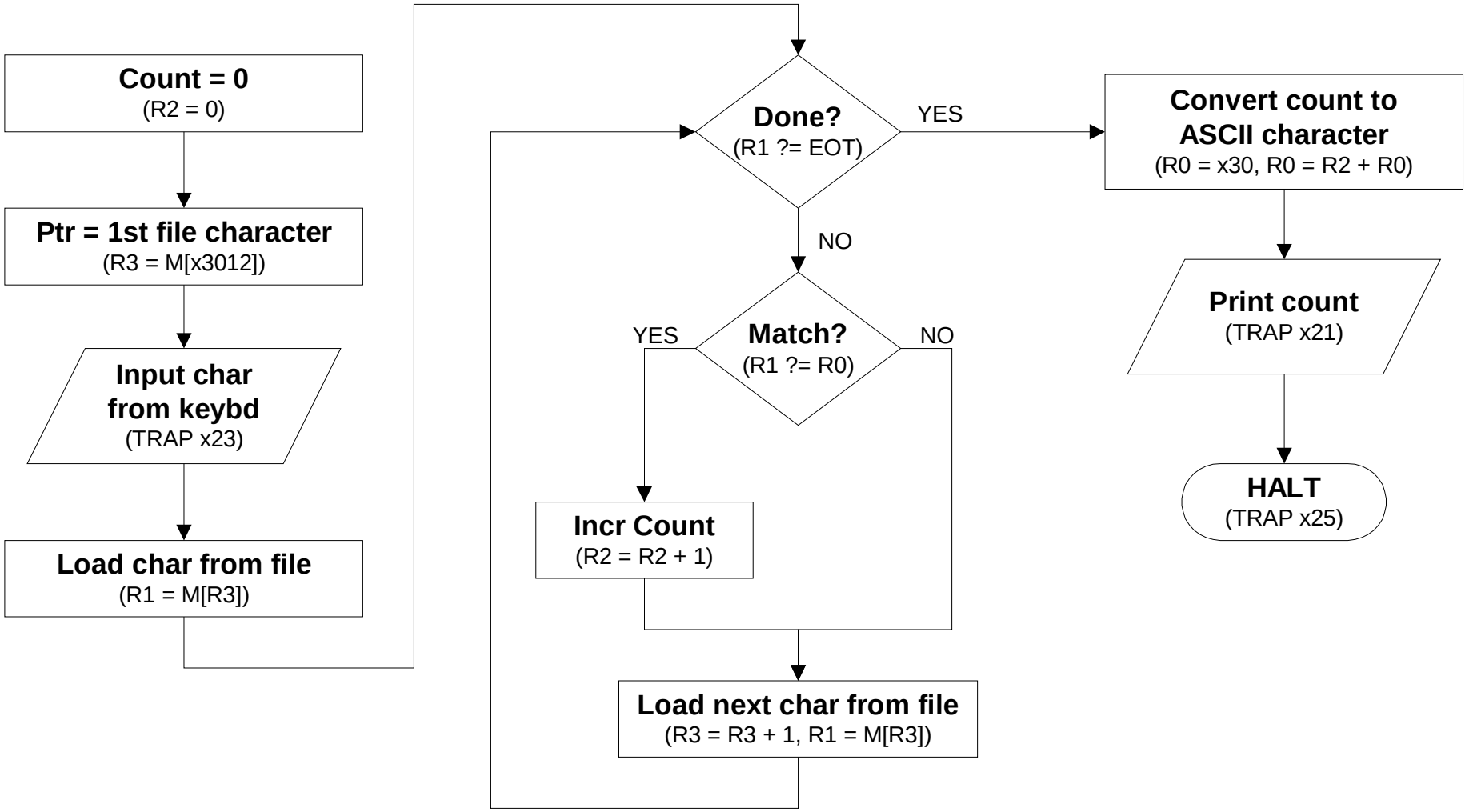
x3100: 1111 0000 0010 0101

例子：字符数统计

计算一个文件中特定字符出现的次数

- **程序开始地址:　x3000**
- **从键盘读入要统计次数的字符**
- **从文件(“file”)中读取字符**
  - Ø程序中的”文件”概念是指一个连续的内存存储区域
  - Ø文件的开始地址紧邻在程序代码的存储区域之后
- **文件中存在和输入字符相同的字符则计数器+1**
- **文件采用哨兵机制.结束标志为 一个特殊的ASCII码: EOT (x04)**
- **程序结束输出统计结果**
  (假定字符出现次数不超过10个, 方便输出)

Flow Chart


Count = 0
(R2 = 0)
Ptr = 1st file character
(R3 = M[x3012])
Input char from keybd
(TRAP x23)
Load char from file
(R1 = M[R3])
Done?
(R1 ?= EOT)
YES
NO
Match?
(R1 ?= R0)
YES
NO
Incr Count
(R2 = R2 + 1)
Load next char from file
(R3 = R3 + 1, R1 = M[R3])
Convert count to ASCII character
(R0 = x30, R0 = R2 + R0)
Print count
(TRAP x21)
HALT
(TRAP x25)

Program (1 of 2)

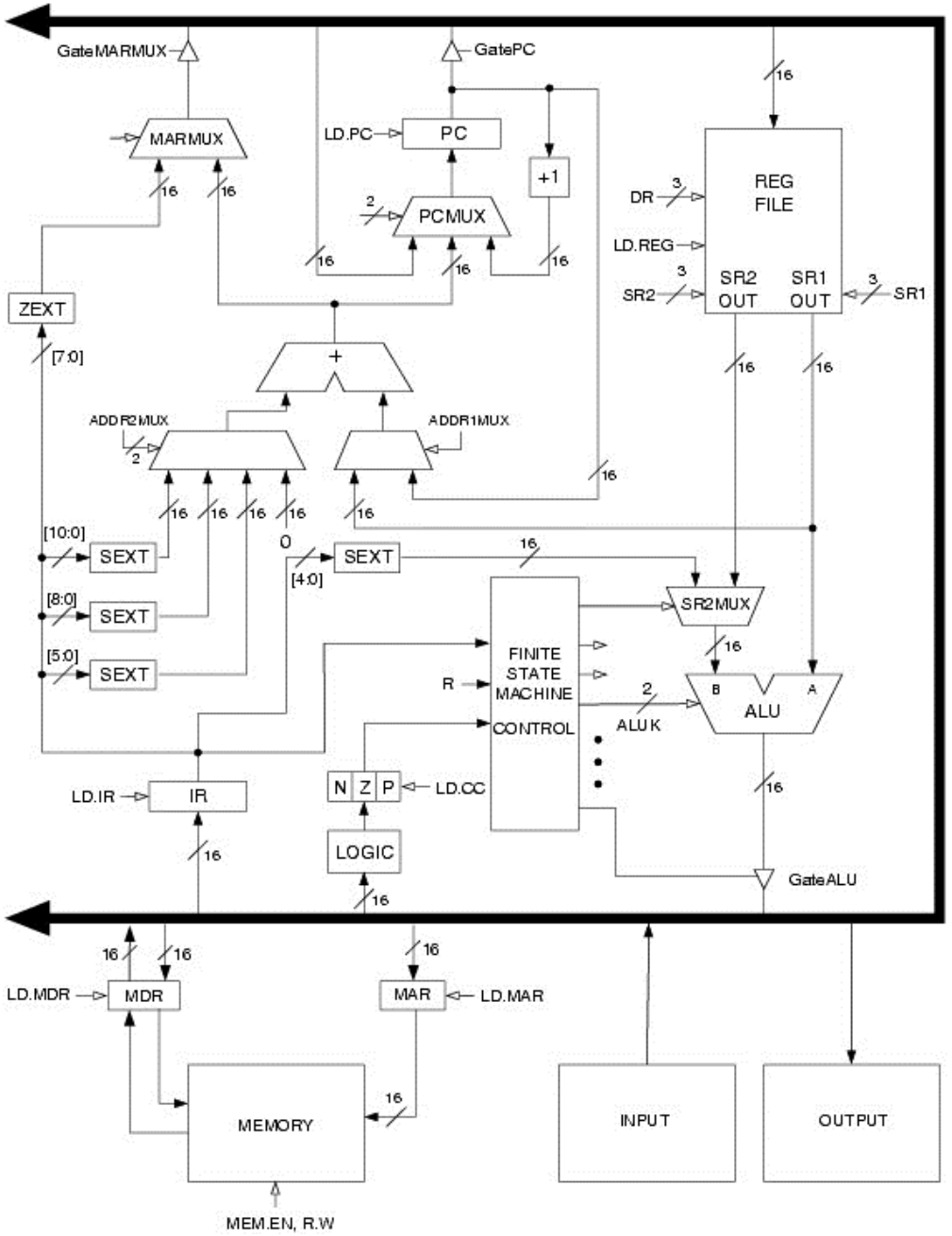
Address	*Instruction*	*Comments*
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ¬ 0 (counter)*
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0	*R3 ¬ M[x3102] (ptr)*
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	*Input to R0 (TRAP x23)*
x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ¬ M[R3]*
x3004	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0	*R4 ¬ 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 ¬ NOT R1*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*R1 ¬ R1 + 1*
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	*R1 ¬ 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)

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

LC-3
总结：数据通路

实心箭头：待处理信息
空心箭头：控制信号



数据通路的基本部件

全局总线

- 一组16位信号线：用于部件之间数据通信
- 只有一个总线，多个部件用。
- 总线连接的输入设备为“三态设备”
  只有被允许时才能使用总线进行传输, 平时为悬浮态。
- 任何时刻只有一个输入被“使能”，即被允许使用总线
  - Ø由控制器统一决定当前哪个输入“使能”
- 任何部件都能读取总线的信息。
  - Ø寄存器仅当写操作时被控制器允许从总线获取数据

内存空间

- I/O设备的控制和数据寄存器
- 内存访问寄存器MAR, MDR 以及读写控制信号

数据通路的基本部件

ALU

- 输入来自寄存器组和指令的立即数（符号扩展到16位）

- 输出结果送到总线.
  - Ø结果将送到寄存器，内存或送到
  - Ø状态码逻辑电路产生N Z P标志位

寄存器组

- 两个读地址(SR1, SR2),一个写地址 (DR)，可以相同
- 输入来自总线（DR）
  - ØALU 算术运算和内存读的结果
- 两个16位的输出
  - Ø作为 ALU的输入, PC的改写值, 间接访问的内存地址
  - Ødata for store instructions passes through ALU

数据通路的基本部件

PC、PCMUX

- PC三类输入来源，由PCMUX控制
  - Ø PC+1 – 取址阶段
  - Ø 地址加法器 – BR, JMP
  - Ø 总线 – TRAP (后续课程)

MAR、MARMUX

- MAR两类输入来源，由MARMUX控制
  - Ø 地址加法器： – LD/ST, LDR/STR
  - Ø IR[7:0]无符号扩展 – TRAP

数据通路的基本部件

条件寄存器

- 查看总线数据，设置**N, Z, P**
- 需**LD.CC**信号开通后，才进行设置，与指令有关(ADD, AND, NOT, LD, LDI, LDR, LEA)

控制单元 – 有限状态机

- 每一个机器周期，发送相关控制信号，包括：
  - **Ø**总线控制信号(GatePC, GateALU, …)
  - **Ø**寄存器控制信号(LD.IR, LD.REG, …)
  - **ØALU**控制信号**?** (ALUK)
  - **Ø…**
- **Logic**：负责**opcode**译码等

作业

Ex 5.4, 5.6, 5.8, 5.9, 5.11, 5.12

Ex 5.16, 5.30, 5.32, 5.33, 5.40

Ex 5.13, 5.14, 5.15, 5.22, 5.23, 5.25