ICS 复习笔记

2023 年 1 月 17 日

目录

1 bit、数据类型及其运算

1.1 浮点数

IEEE-32 位浮点数:
符号位:1bit, 代表符号 (正数或负数)
数值范围:8bit, 代表范围 (指数:exponent, 无符号整数)
数值精度:23bit, 代表精度 (尾数:fraction, 无符号整数)

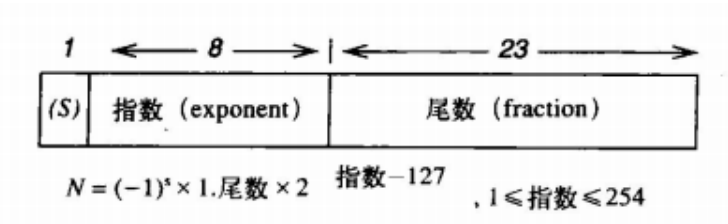


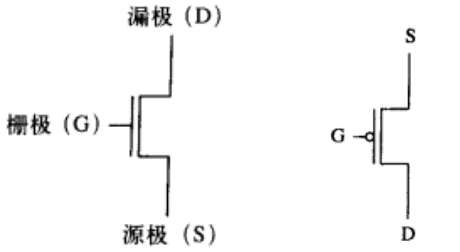
图 1: IEEE-32 位浮点数

若指数字段是 00000000 表示指数为-126 但小数点前一位是 0(不是 1)
若指数字段是 11111111, 尾数字段全为 0 则表示无穷
若指数字段是 11111111, 尾数字段不全为 0 则表示 NaN

2 数字逻辑

2.1 MOS 晶体管

n-MOS 管: 高压通, 低压断 p-MOS 管: 高压断, 低压通

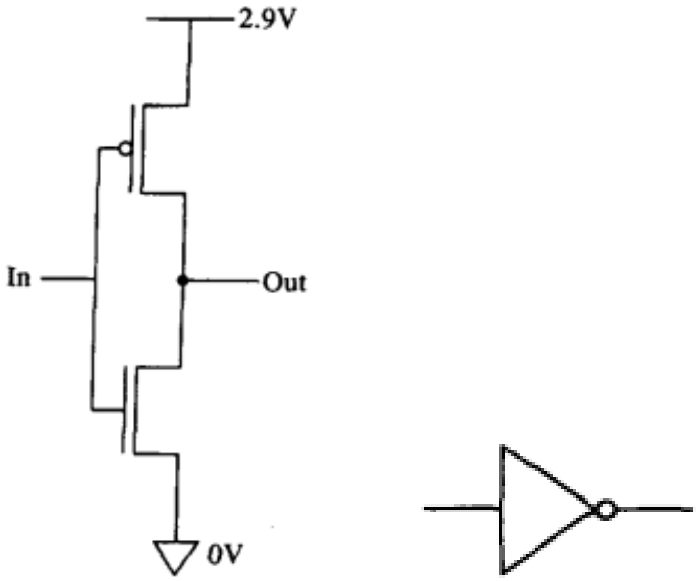


(a) n-MOS 管 (b) p-MOS 管

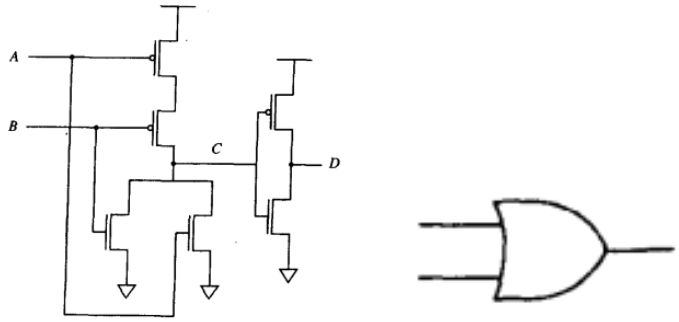
图 2: MOS 管

2.2 逻辑门

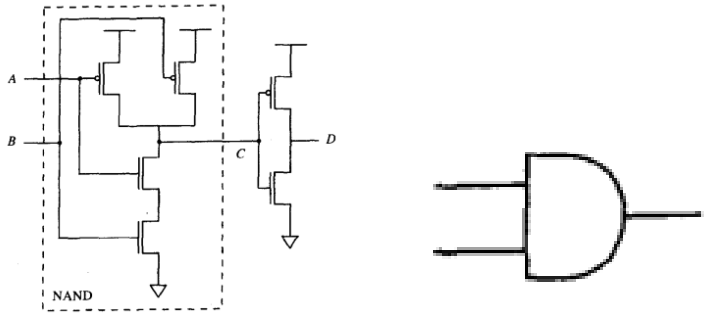
1. 非门 (NOT):



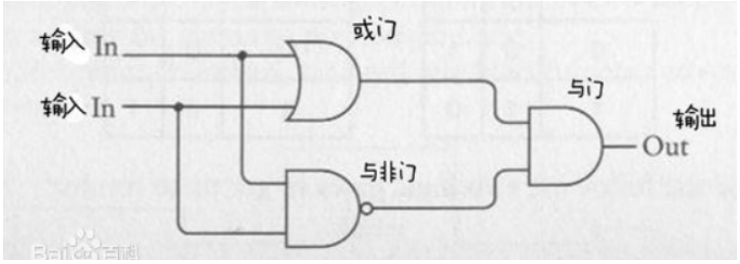
2. 或门 (OR):



3. 与门 (AND):



4. 异或门 (XOR):



2.3 组合逻辑

1. 译码器 (decode): 所有输出中有且仅有一个为 1, 用于检测、匹配不同的输入模式

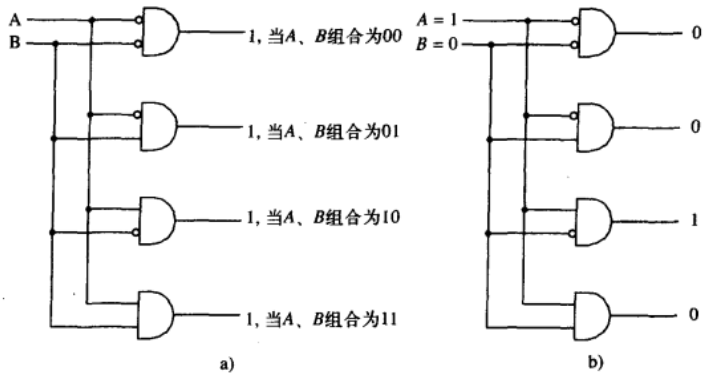


图 3: 2-输入译码器

2. 多路复用器 (MUX): 从多个输入中选择一个, 并将其与输出相连, 选择信号 S 负责决定究竟选择哪一个

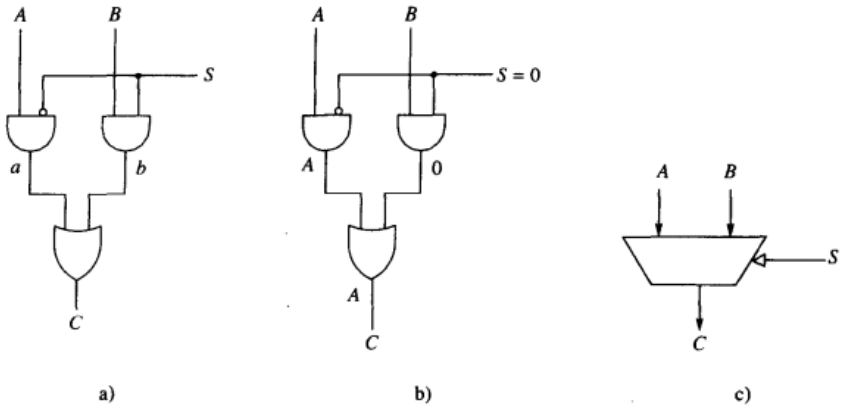


图3-12 2选1多路复用器

3. 全加器:A、B 表示当前位,C 表示进位,S 表示当前求和位

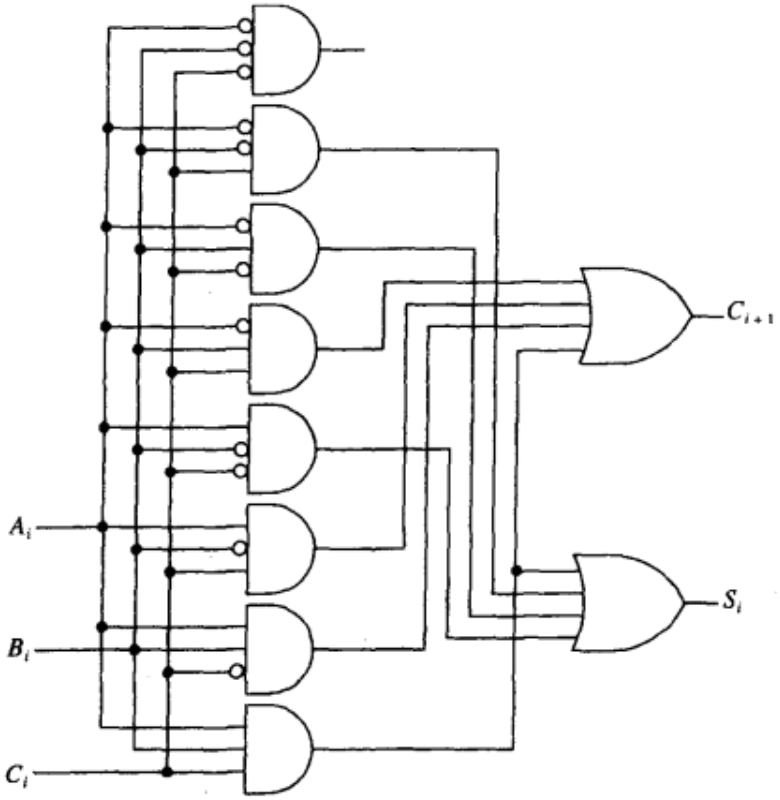
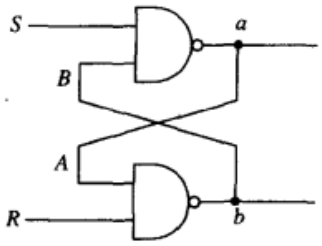


图3-15 全加器的门电路

2.4 存储单元

4.R-S 锁存器:

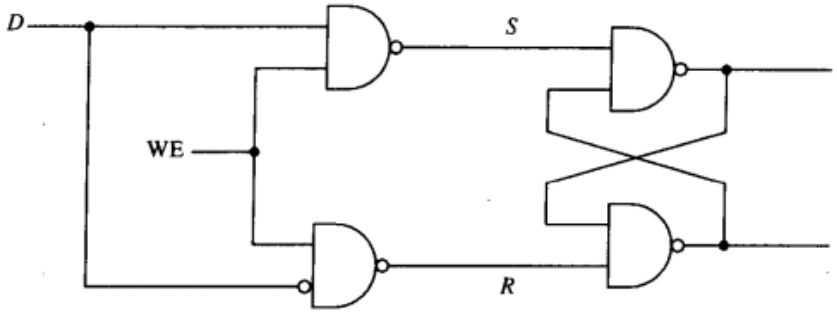


只要输入 R、S 的值都为 1 则输出 a 的值不变

清除 S, 则输出 a 为 1, 清除 R, 则输出 a 为 0;

不要同时把 R、S 清零

5. 门控 D 锁存器:



若 WE=1 表示“可写”, 此时 R-S 锁存器的输出 =D

若 WE=0, 此时 R、S 都为 1, 锁存器保持静态

6. 寄存器: 一个 16bit 寄存器, 从右向左递增, 最左边为 bit[n-1], 用 Q[l:r] 表示从 bit[l] 到 bit[r] 的一个字段

2.5 内存

寻址空间: 内存中可独立识别的位置总数

寻址能力: 每个内存位置包含的 bit 数

eg.$2^2 \times 3$ 内存, 表示寻址空间为 2^2, 寻址能力为 3 的内存

2.6 时序电路

1. 有限状态机:

(1) 状态 (有限数目)

(2) 外部输入 (有限数目)

(3) 对外输出 (有限数目)

(4) 任意状态间迁移 (显式注明)

(5) 对外输出操作 (显式注明)

2. 时钟:0、1 交替变换的信号

时钟周期: 时钟信号不断变换的间隔时间

在电路实现中, 有限状态机的状态转移发生在每个时钟周期的起始时刻

2.7 LC-3 的数据通路

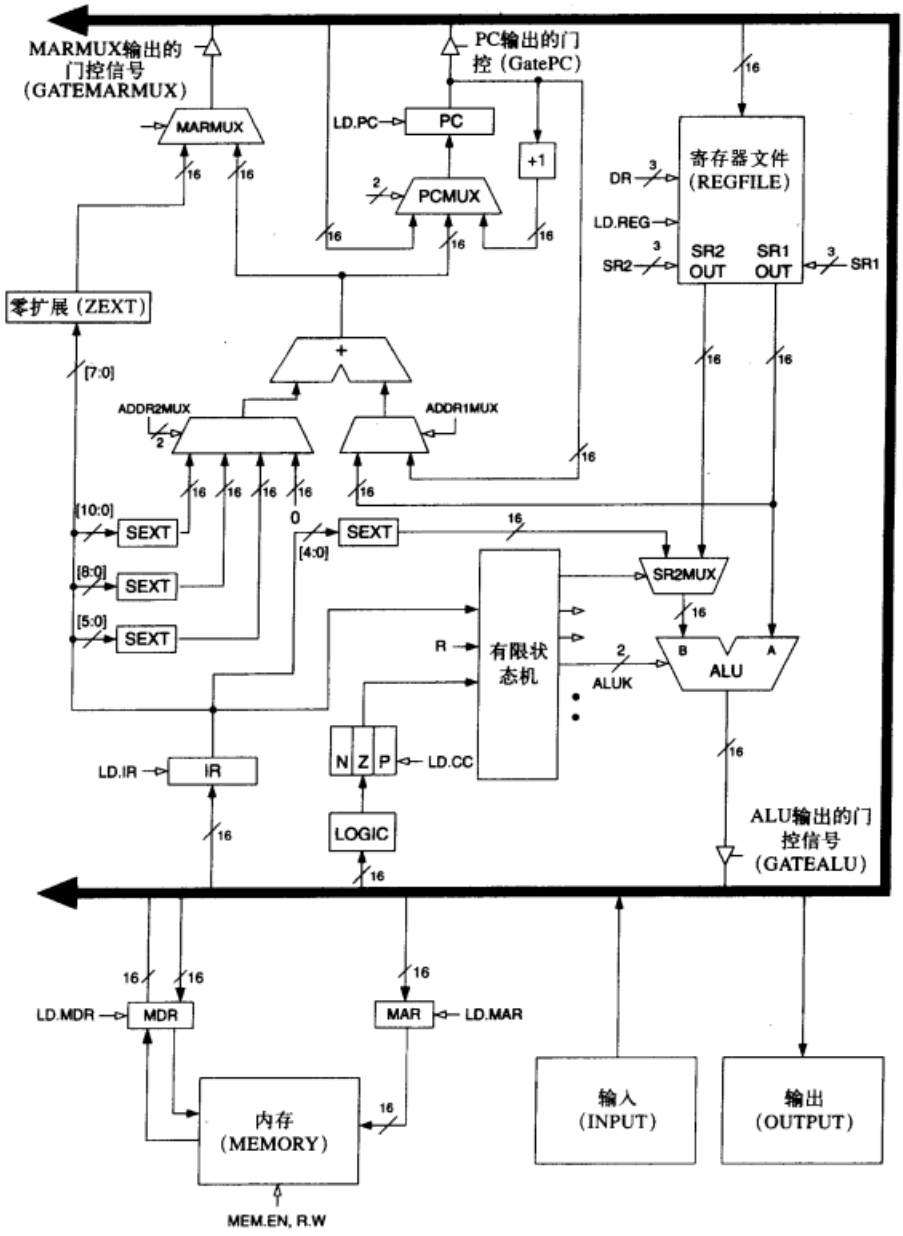


图 4: LC-3 的数据通路

3 冯 · 诺依曼模型

3.1 基本架构

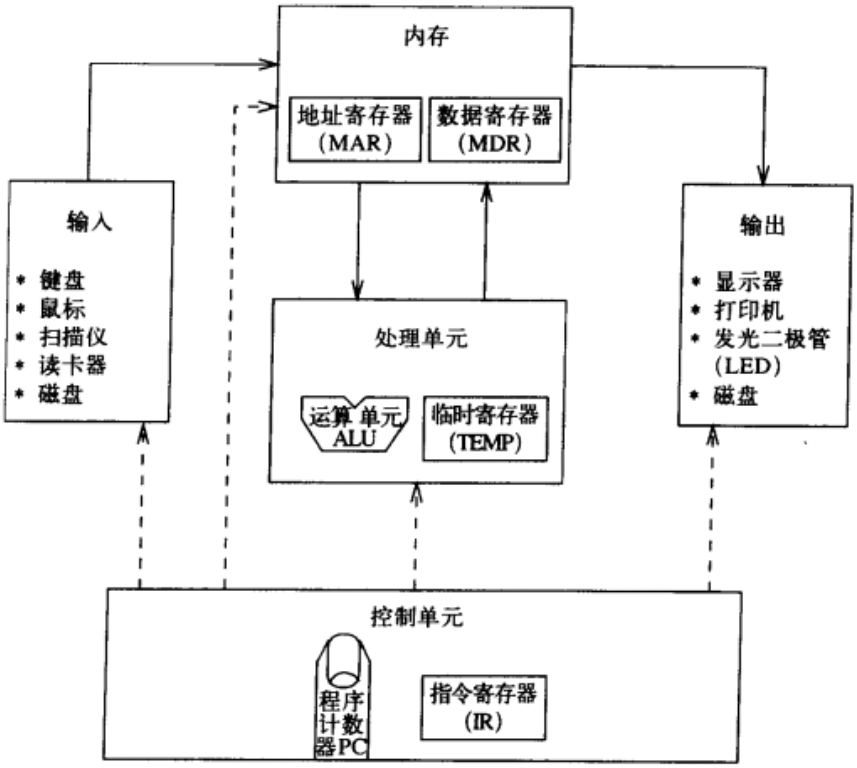


图 5: 冯 · 诺依曼模型

1. 内存

读操作: 将被访问内存单元的地址放入内存地址寄存器 (MAR), 然后发送读信号通知内存, 内存将该单元存储的数据放入内存数据寄存器 (MDR)

写操作: 将被访问内存单元的地址放入内存地址寄存器 (MAR), 然后将要写入的数据放入内存数据寄存器 (MDR), 最后向内存发送写信号

2. 处理单元

算数逻辑运算单元 (ALU)

ALU 能处理的量化大小被称为“字长”, 如 LC-3 的字长是 16 位

通用寄存器 ($R0 \sim R7$):ALU 的临时储存空间

3. 输入/输出单元
输入设备: 键盘
输出设备: 显示器
4. 控制单元指令寄存器 (IR): 保存正在被执行的指令
指令指针 (PC): 保存下一条指令的地址

3.2 指令处理

1. 指令结构
一条 16bit 的指令:bit[15:12] 表示操作码,bit[11:0] 表示操作数
2. 指令周期
(1)Fetch(取指令):(3 machine cycle)

- 将 PC 寄存器的内容装入 (load)MAR 寄存器
- PC 增量 (1 machine cycle)
- 该地址对应内存单元的内容 (即下一条指令) 被装入 MDR(1 or more machine cycle)
- 控制单元将 MDR 的内容装入 IR 寄存器 (1 machine cycle)

(2)Decode(译码):(1 machine cycle)
使用译码器分析指令的操作码, 每一根使能线对应一种指令
(3)Evaluate Address(地址计算):
如果指令执行时存在地址计算操作, 则在此节拍完成
(4)Fetch operand(取操作数):
读取指令所需的源操作数
(5)Execute(执行):
执行指令
(6)Store result(存放结果):
执行结果被写入目的寄存器

3.3 状态控制

1. 有限状态机控制指令周期的执行
2. 停机操作

用户程序由操作系统停止
整体停机需把时钟电路的 RUN 锁存器清零

4 LC-3 结构

4.1 概述

1. 内存组织:

   寻址空间:2^{16}, 寻址能力:16, 字长:16

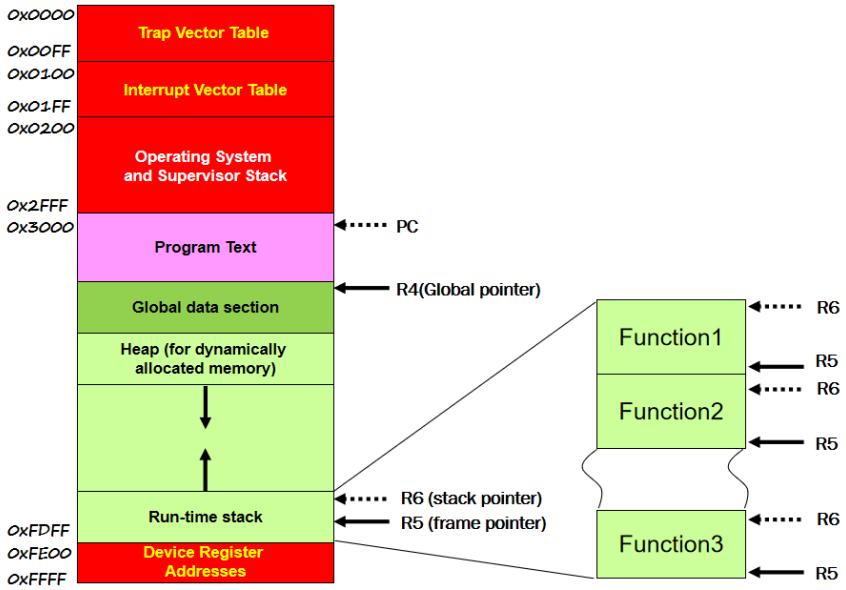


图 6: LC-3 的内存组织

2. 寄存器:

   8 个通用寄存器 (GPR):$R0 \sim R7$

   3 个位寄存器 (condition code):N、Z、P

3. 指令集:

   一条指令分为两个部分: 操作码 (做什么) 和操作数 (对谁操作), 一个 ISA 的指令集定义包括: 操作码的集合、数据类型和寻址模式, 其中寻址模式决定了操作数的存放位置

4. 操作码:

LC-3 的 ISA 定义了 15 条指令和一条预留指令, 所有指令可以分为三类: 运算 (operate)、数据搬移 (data movement) 和控制 (control), 下面是 LC-3 的所有指令及其格式:

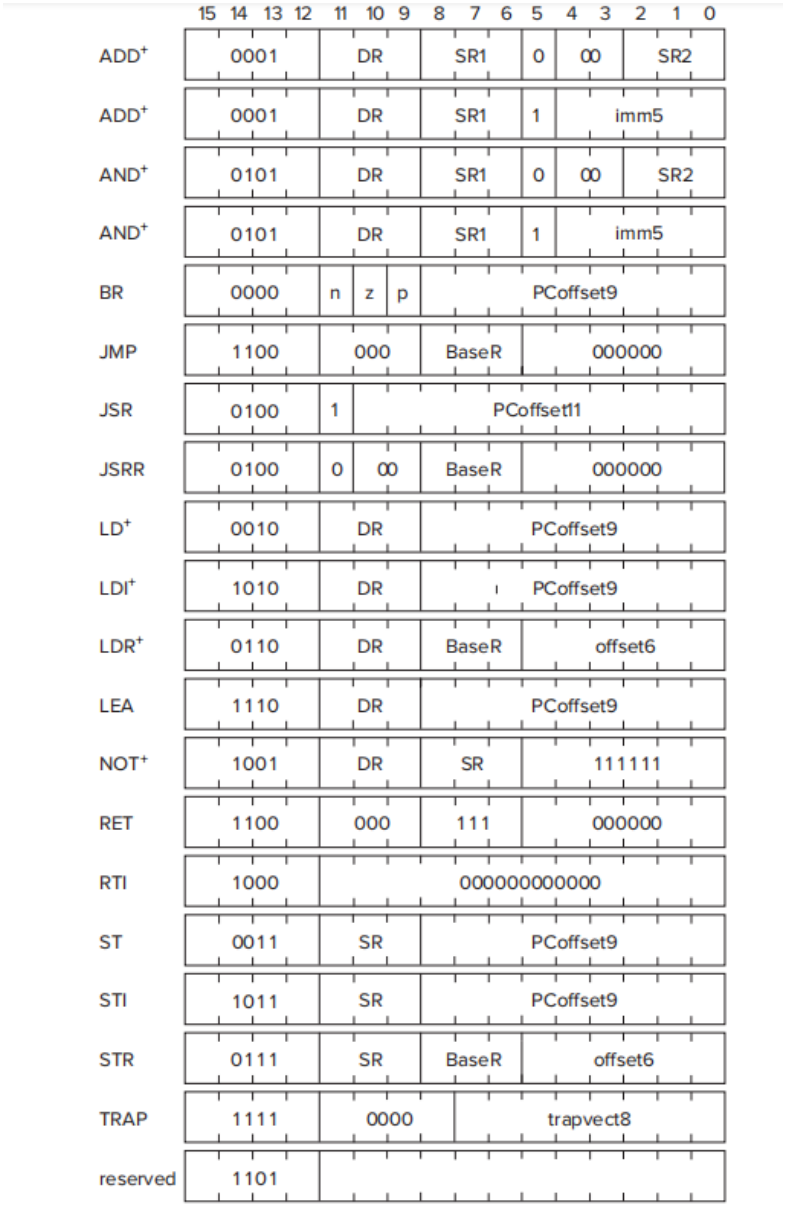


Figure 5.3 Formats of the entire LC-3 instruction set. *Note*: + indicates instructions that modify condition codes.

5. 数据类型:

   LC-3 的 ISA 唯一支持的数据类型: 补码整数

6. 寻址模式:

   LC-3 支持 5 种寻址模式: 立即数、寄存器、PC 相对寻址、间接寻址和基址偏移寻址

7. 条件码:

   N、Z、P 分别对应负数、零和正数, 每当任意 GPR 寄存器被写入时, 根据写入结果是负数、零或正数, 把对应条件码置 1, 其它清零

4.2 ISA

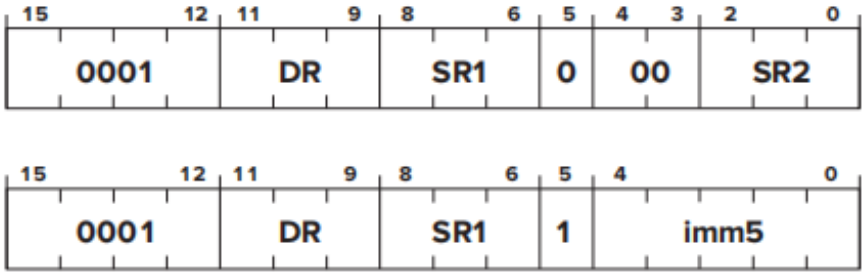
4.2.1 ADD(opcode=0001)

1. 汇编语言:

   ADD DR, SR1, SR2

   ADD DR, SR, imm5

2. 编码:



3. 操作:

```
if(bit[5]==0)
    DR=SR1+SR2;
else
    DR=SR+SEXT(imm5);
setcc();
```

4. 描述:

   如果 bit[5] 是 0, 则第二个操作数来自 SR2; 如果 bit[5] 是 1, 则第二个操作数来自 imm5 的符号扩展. 无论操作数来自哪, 都将与 SR1 相加, 并将结果存入 DR, 然后修改条件码。

5. 数据通路:

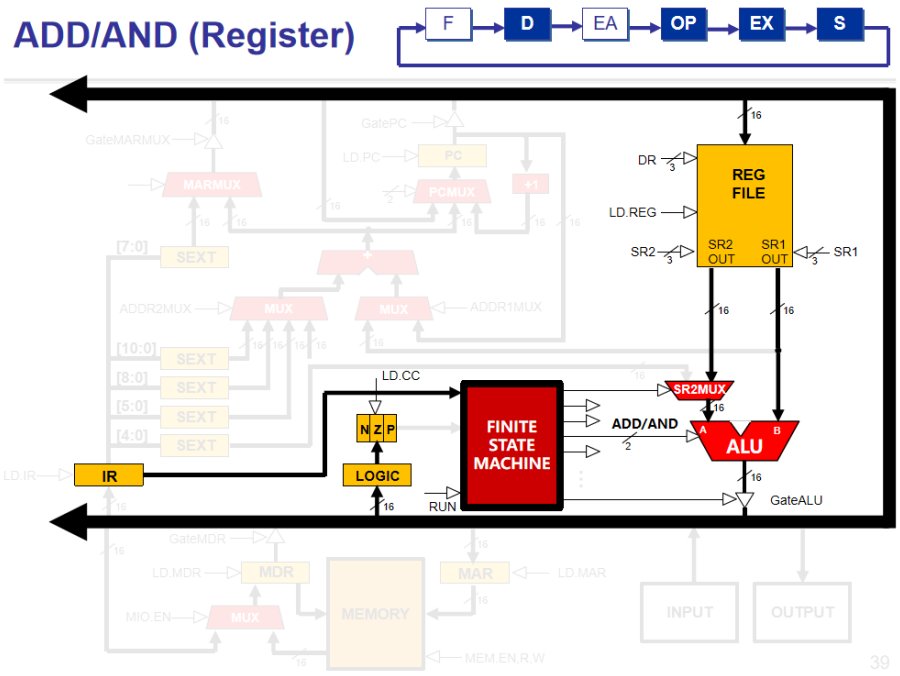


图 7: 数据通路-ADD 和 AND

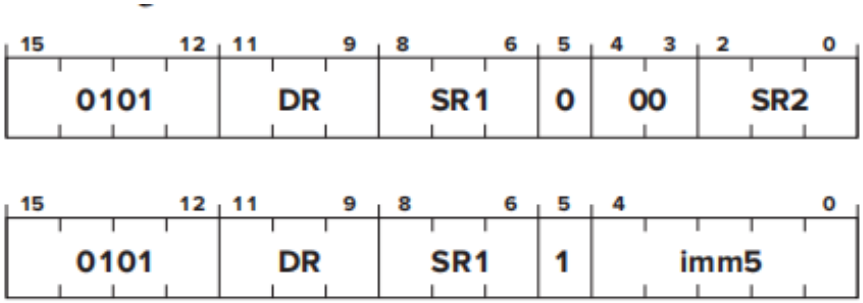
4.2.2 AND(opcode=0101)

1. 汇编语言:

   AND DR, SR1, SR2

   AND DR, SR, imm5

2. 编码:



3. 操作:

```
if(bit[5]==0)
    DR=SR1 AND SR2;
else
    DR=SR  AND SEXT(imm5);
setcc();
```

4. 描述:

   如果 bit[5] 是 0, 则第二个操作数来自 SR2; 如果 bit[5] 是 1, 则第二个操作数来自 imm5 的符号扩展. 无论操作数来自哪, 都将与 SR1 做按位” 与” 运算, 并将结果存入 DR, 然后修改条件码。

5. 数据通路:

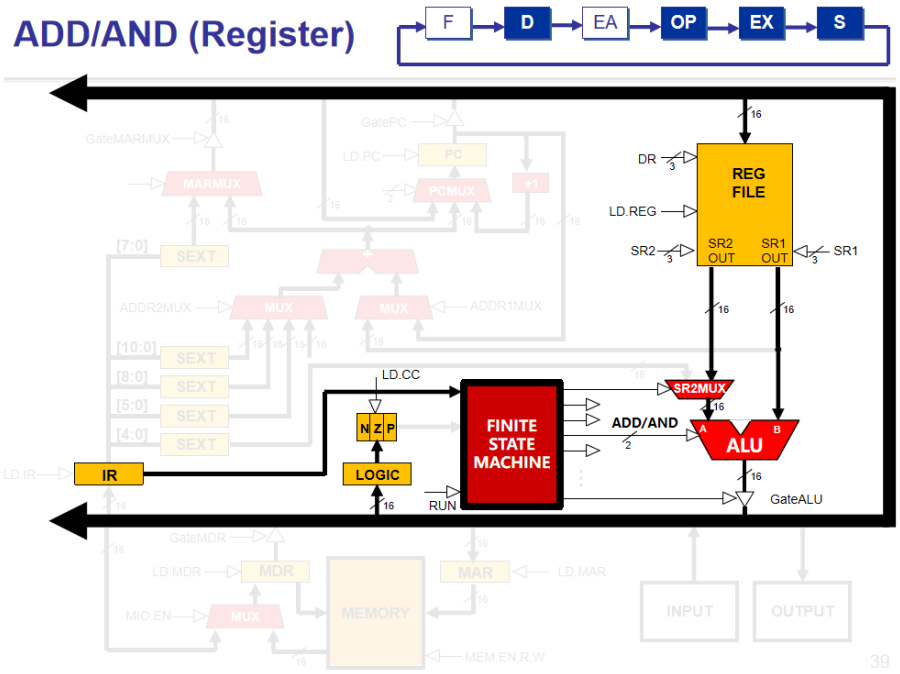


图 8: 数据通路-ADD 和 AND

4.2.3 BR(opcode=0000)

1. 汇编语言:

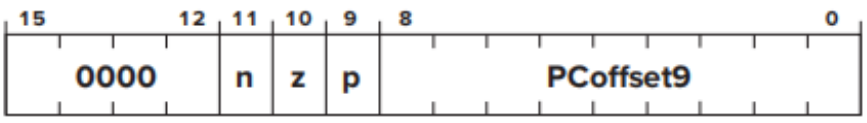
   BRn LABEL BRzp LABEL

   BRz LABEL BRnp LABEL

   BRp LABEL BRnz LABEL

   BR LABEL BRnzp LABEL

2. 编码:



3. 操作:

```
1    if((n&&N)||(z&&Z)||(p&&P))
2        PC=PC+1+SEXT(PCoffset9);
```

4. 描述:

   由 bit[11:9] 指定的条件码被测试 (如 bit[11]=1, 则 N 被测试, 否则不测试 N), 如果任何一个被指定测试的条件码被置位, 则程序跳转到 PC 增量与 SEXT(PCoffset9) 之和处

5. 数据通路:

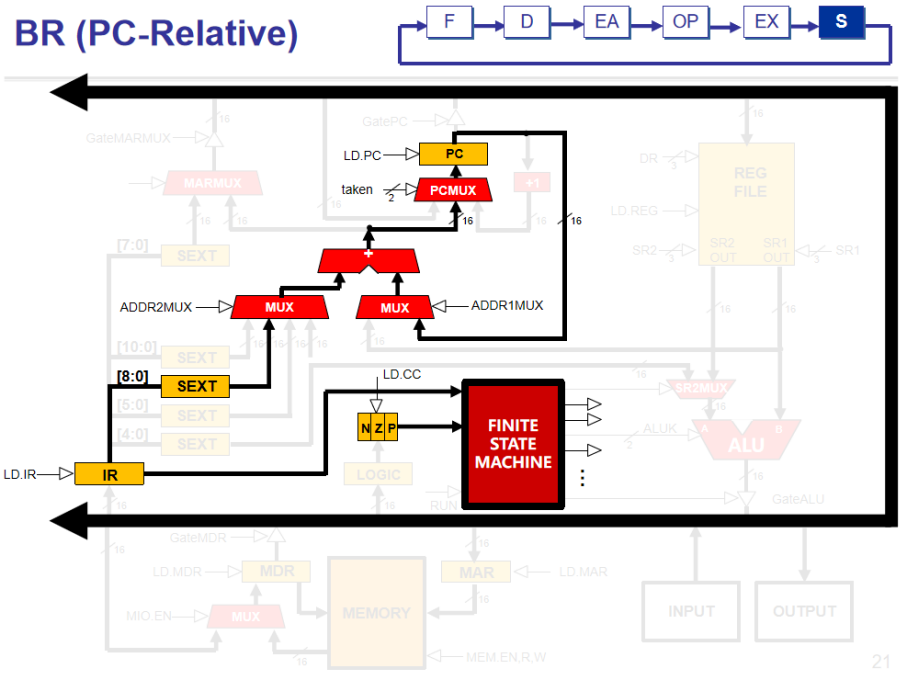


图 9: 数据通路-BR

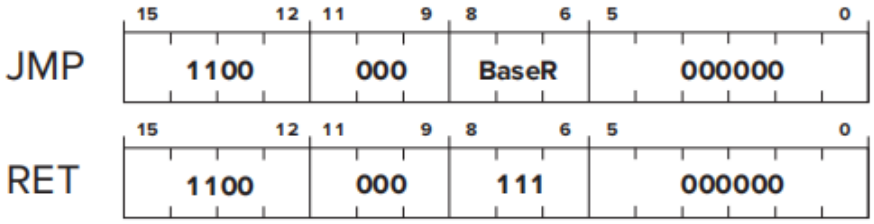
4.2.4 JMP/RET(opcode=1100)

1. 汇编语言:

   JMP BaseR

   RET

2. 编码:



3. 操作:

```
1                    PC=BaseR;
```

4. 描述:

   程序无条件跳转至由基址寄存器指定的入口,bit[8:6] 代表基址寄存器的编号

5. 数据通路:

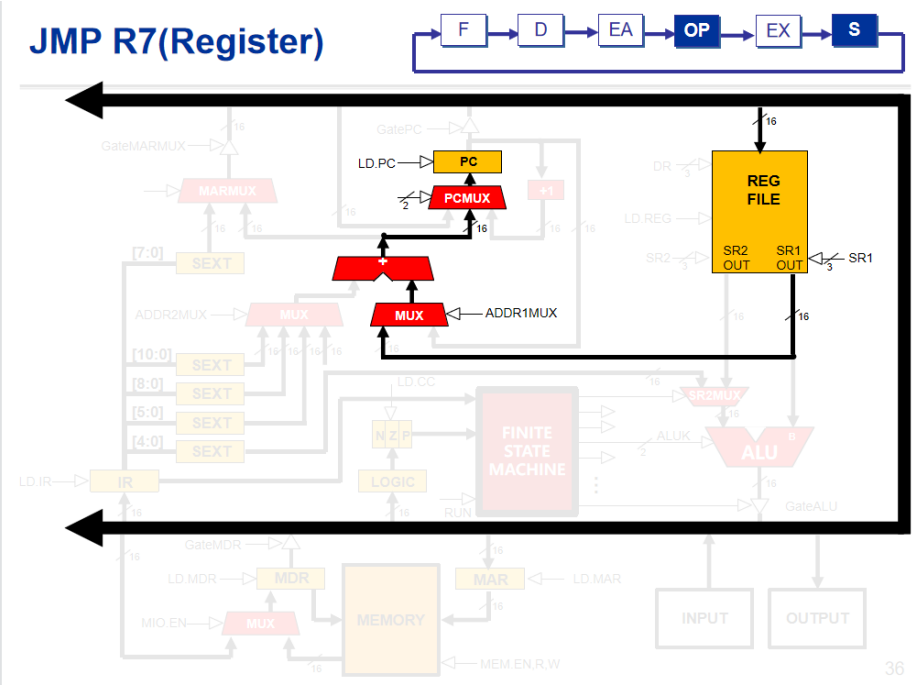


图 10: 数据通路-JMP 和 RET

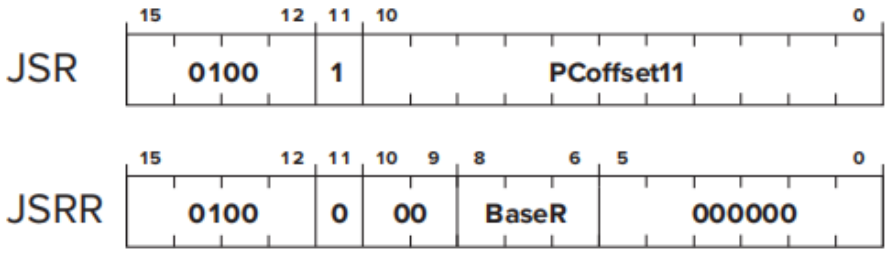
4.2.5 JSR/JSRR(opcode=0100)

1. 汇编语言:

   JSR LABEL

   JSRR BaseR

2. 编码:



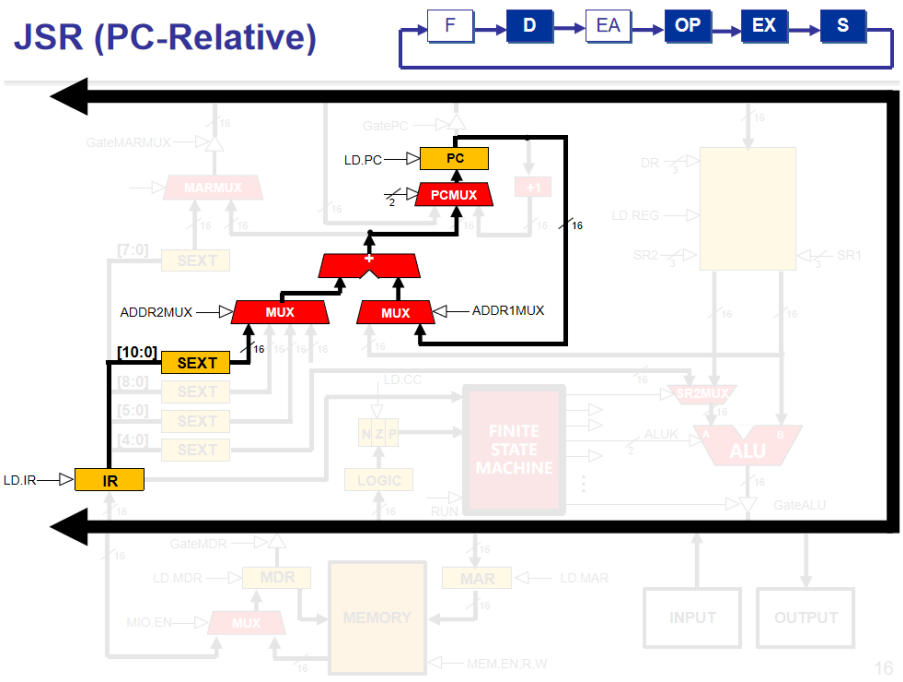
3. 操作:

```
temp=PC;
if(bit[11]==0)
    PC=BaseR;
else
    PC=PC+1+SEXT(PCoffset11);
R7=temp;
```

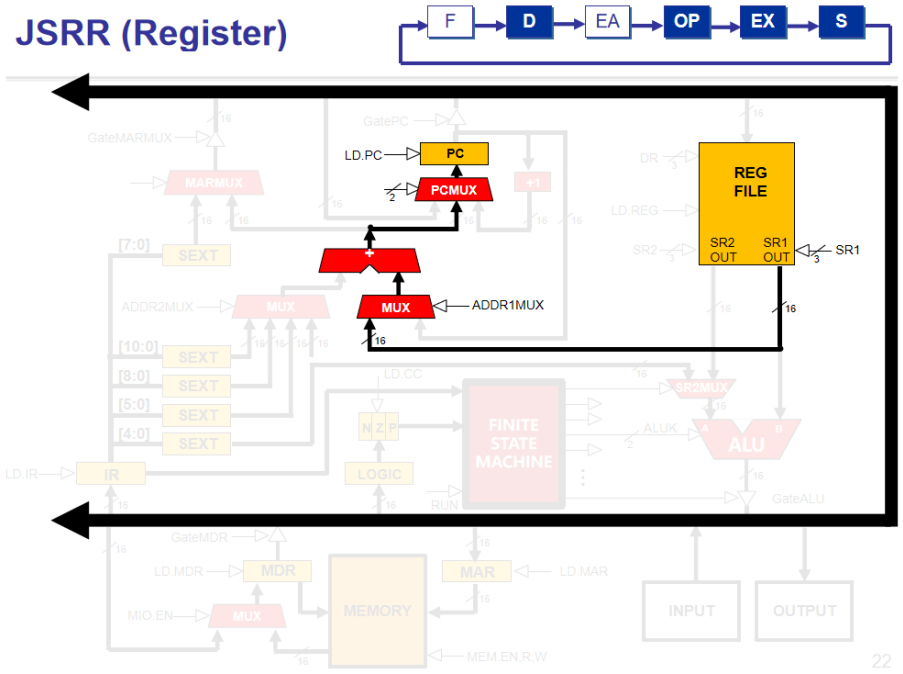
4. 描述:

   首先把当前 PC 的 1 值存入 R7, 用于返回跳转, 然后 PC 装入被调用子程序的入口地址, 这个入口地址来自 BaseR 或 PC 增量与 SEXT(PCoffset11) 之和

5. 数据通路:



(a) 数据通路-JSR(保存 PC 涉及寄存器文件)



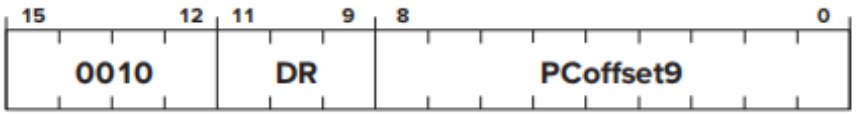
(b) 数据通路-JSRR

4.2.6 LD(opcode=0010)

1. 汇编语言:

   LD DR, LABEL

2. 编码:



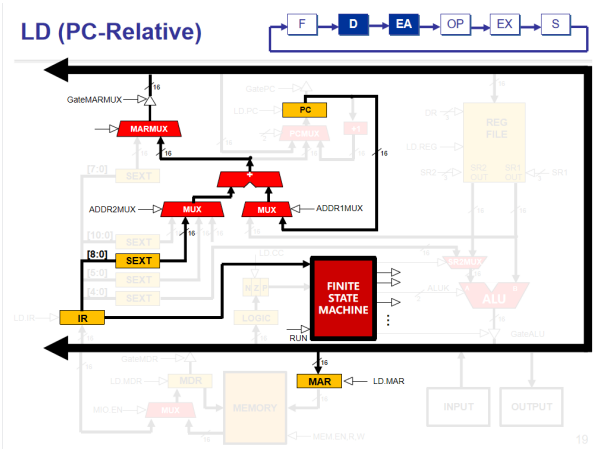
3. 操作:

```
            if ((Computed address is in privileged
memory)&&(PSR[15]==1))
                Initiate ACV exception;
            else
                DR=mem[PC+1+SEXT(PCoffset9)];
            setcc();
```

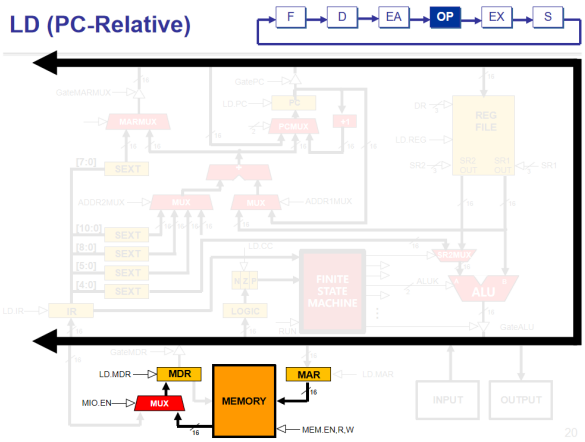
4. 描述:

   由 PC 增量和 SEXT(PCoffset9) 相加得到地址, 若地址是特权内存且 PSR[15]=1, 则启动 ACV 异常, 否则把该地址存储的内容存入 DR, 然后修改条件码

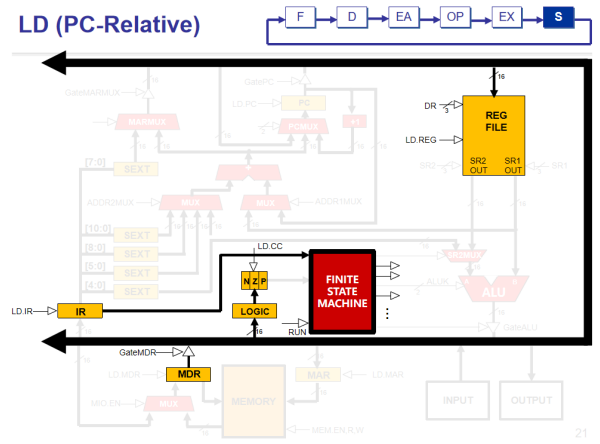
5. 数据通路:



(c) 数据通路-LD 译码和地址计算



(d) 数据通路-LD 取操作数



(e) 数据通路-LD 存结果

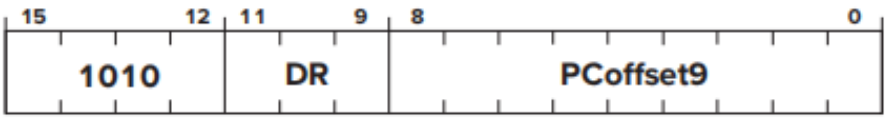
图 11: 数据通路-LD

4.2.7 LDI(opcode=1010)

1. 汇编语言:

   LDI DR, LABEL

2. 编码:



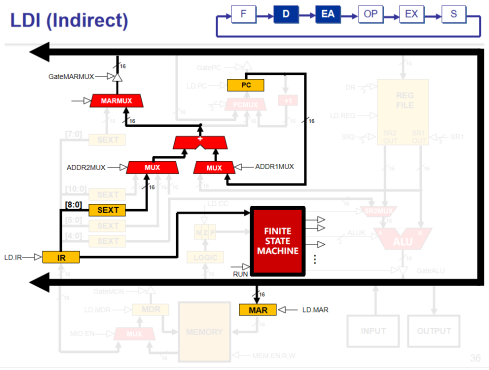
3. 操作:

```
            if ((Computed address is in privileged
memory)&&(PSR[15]==1))
                Initiate ACV exception;
            else
                DR=mem[mem[PC+1+SEXT(PCoffset9)]];
            setcc();
```

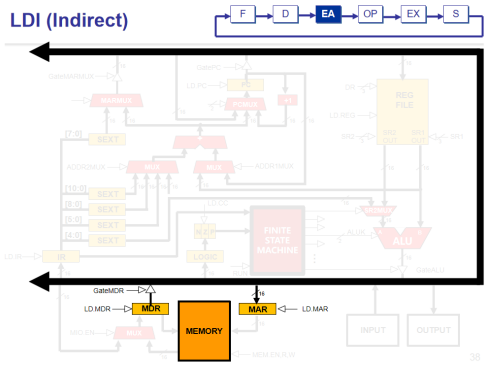
4. 描述:

   由 PC 增量和 SEXT(PCoffset9) 相加得到地址以该地址的内容作为地址, 若地址是特权内存且 PSR[15]=1, 则启动 ACV 异常, 否则把该地址存储的地址的内容存入 DR, 然后修改条件码

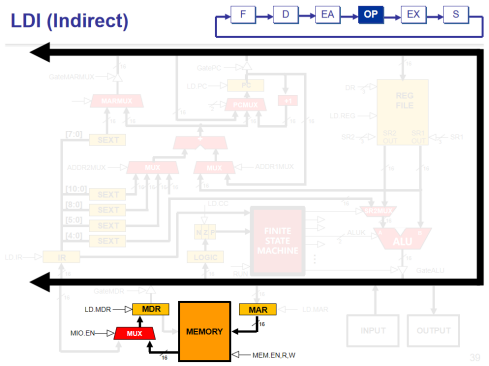
5. 数据通路:



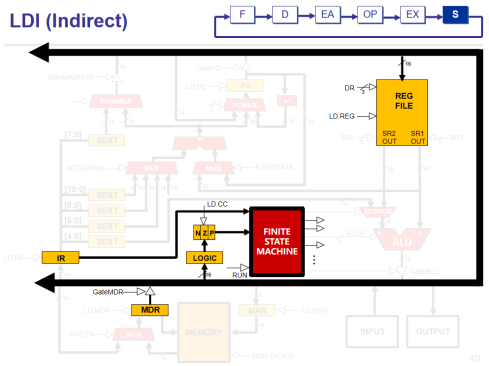
(a) 数据通路-LDI 译码和地址计算



(b) 数据通路-LDI 取操作数后再次计算地址



(c) 数据通路-LDI 再取操作数



(d) 数据通路-LDI 存结果

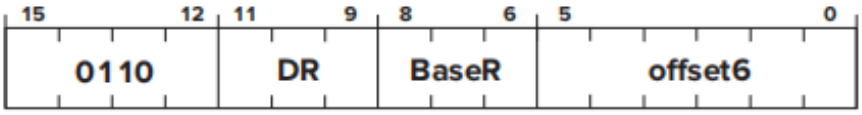
图 12: 数据通路-LDI

4.2.8 LDR(opcode=0110)

1. 汇编语言:

   LDR DR, BaseR, offset6

2. 编码:



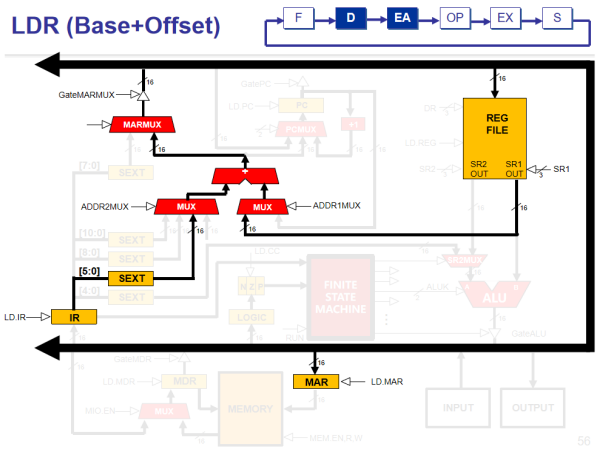
3. 操作:

```
            if ((Computed address is in privileged
memory)&&(PSR[15]==1))
                Initiate ACV exception;
            else
                DR=mem[BaseR+SEXT(PCoffset6)];
            setcc();
```

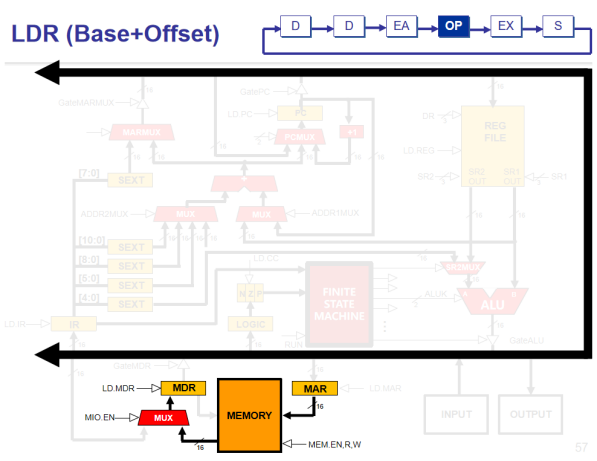
4. 描述:

   由 BaseR 的内容和 SEXT(offset6) 相加得到地址, 若地址是特权内存且 PSR[15]=1, 则启动 ACV 异常, 否则把该地址存储的内容存入 DR, 然后修改条件码

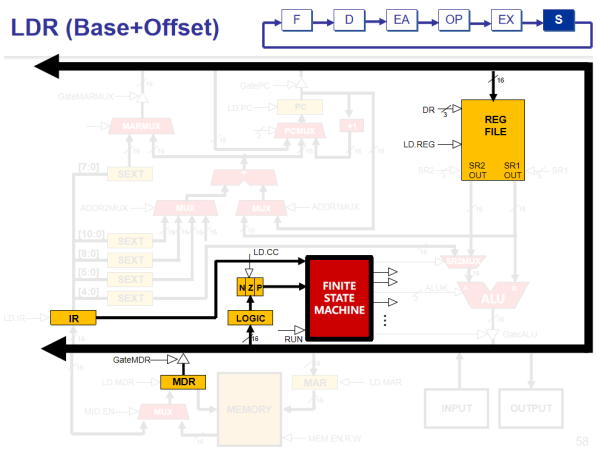
5. 数据通路:



(a) 数据通路-LDR 译码和地址计算



(b) 数据通路-LDR 取操作数



(c) 数据通路-LDR 存结果

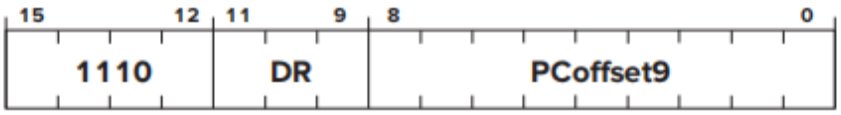
图 13: 数据通路-LDR

4.2.9 LEA(opcode=1110)

1. 汇编语言:

   LEA DR, LABEL

2. 编码:



3. 操作:

```
1    DR=PC+1+SEXT(PCoffset9);
```

4. 描述:

   把 PC 增量与 SEXT(PCoffset9) 相加得到地址装入 DR, 不修改条件码

5. 数据通路:

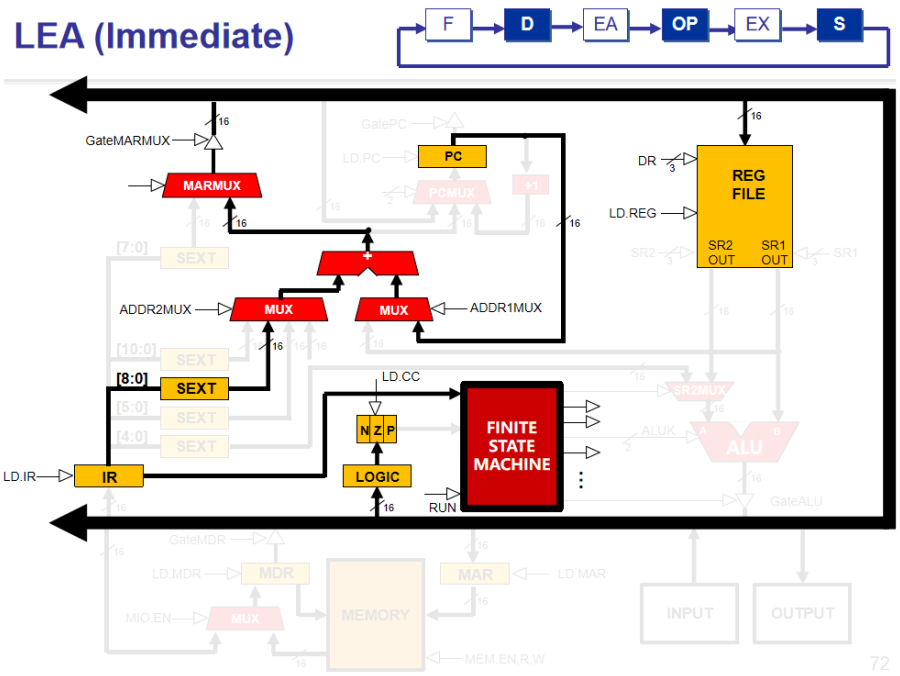


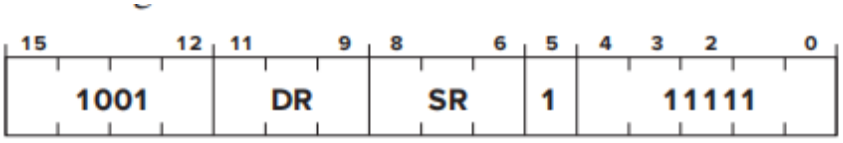
图 14: 数据通路-LEA

4.2.10 NOT(opcode=1001)

1. 汇编语言:

   NOT DR, SR

2. 编码:



3. 操作:

```
1    DR=NOT(SR);
2    setcc();
```

4. 描述:

   把 SR 的内容按位取反存入 DR, 然后修改条件码

5. 数据通路:

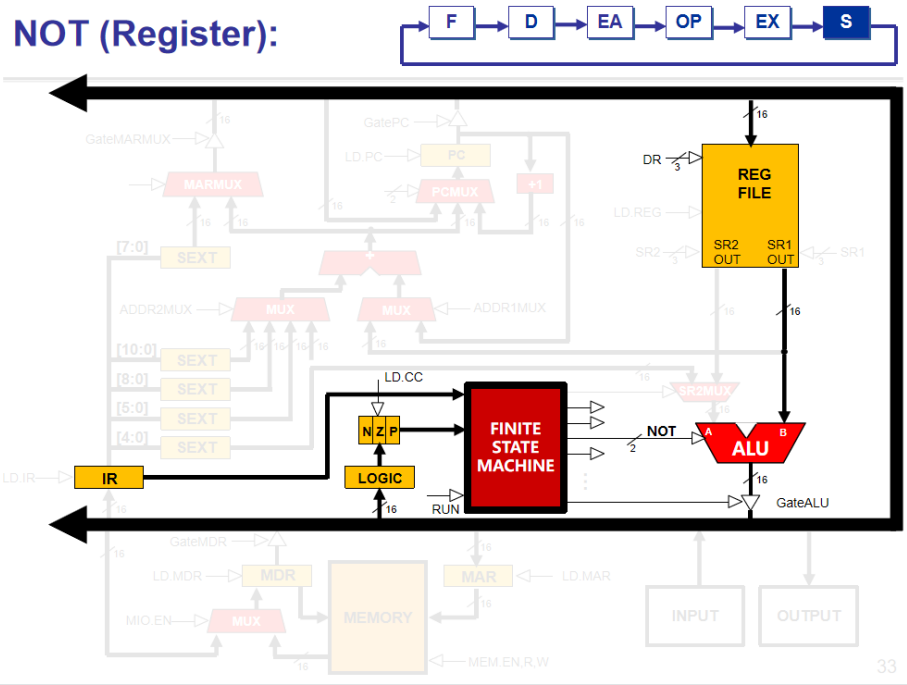


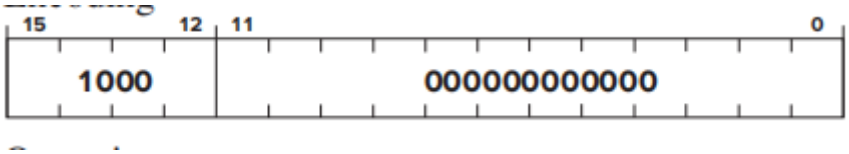
图 15: 数据通路-NOT

4.2.11 RTI(opcode=1000)

1. 汇编语言:

   RTI

2. 编码:



3. 操作:

```
if (PSR[15] == 1)
    Initiate a privilege mode exception;
else
{
    PC=mem[R6]; R6 is the SSP, PC is restored
    R6=R6+1;
    TEMP=mem[R6];
    R6=R6+1; system stack completes POP before saved PSR is
    PSR=TEMP; PSR is restored
    if (PSR[15] == 1)
    {
        Saved SSP=R6;
        R6=Saved USP;
    }
}
```

4. 描述:

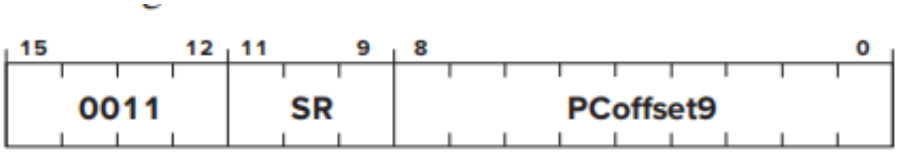
   如果处理器在用户模式下运行, 则会发生特权模式异常. 如果处于特权模式模式, 系统堆栈上的前两个元素将弹出并加载到 PC、PSR 中. 恢复 PSR 后, 如果处理器在用户模式下运行, 则 SSP 保存在保存的 SSP 中,R6 加载保存的 USP。

4.2.12 ST(opcode=0011)

1. 汇编语言:

   ST SR, LABEL

2. 编码:



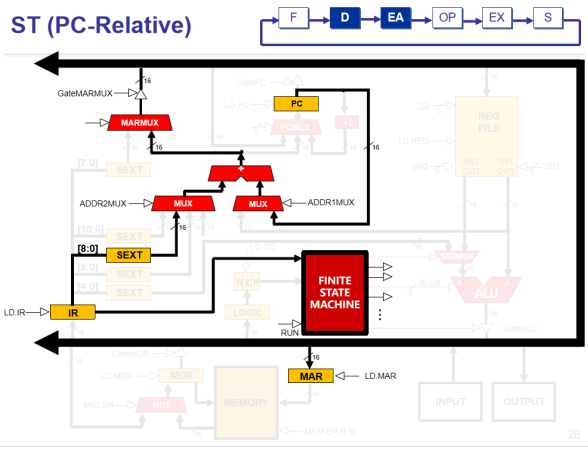
3. 操作:

```
1             if((Computed address is in privileged
2 memory)&&(PSR[15]==1))
3                 Initiate ACV exception;
4             else
5                 mem[PC+1+SEXT(PCoffset9)]=SR;
```

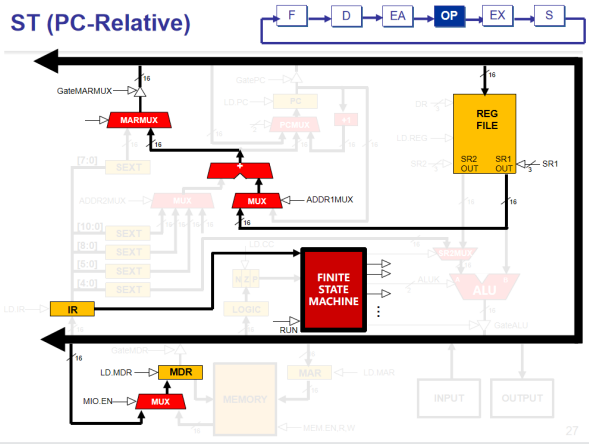
4. 描述:

   由 PC 增量和 SEXT(PCoffset9) 相加得到地址, 若地址是特权内存且 PSR[15]=1, 则启动 ACV 异常, 否则把 SR 存入该地址

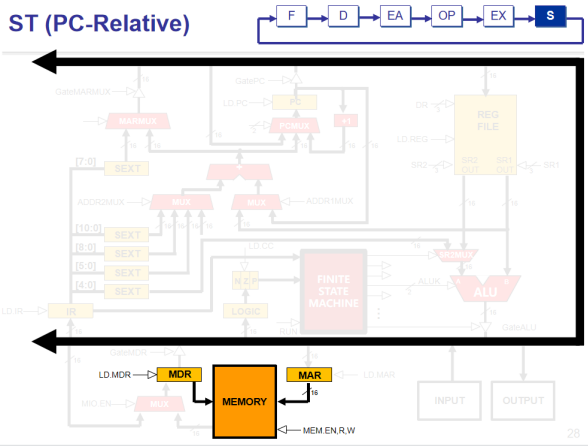
5. 数据通路:



(a) 数据通路-ST 译码和地址计算



(b) 数据通路-ST 取操作数



(c) 数据通路-ST 存结果

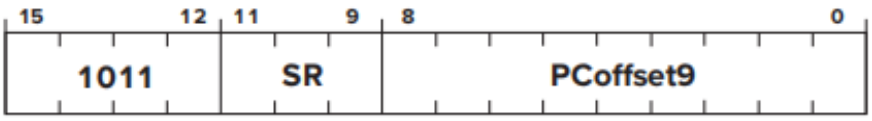
图 16: 数据通路-ST

4.2.13 STI(opcode=1011)

1. 汇编语言:

   STI SR, LABEL

2. 编码:



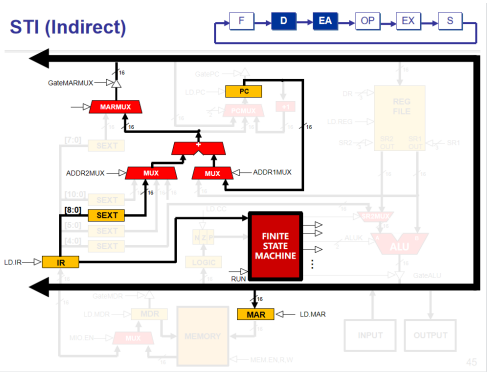
3. 操作:

```
            if ((Computed address is in privileged
memory)&&(PSR[15]==1))
                Initiate ACV exception;
            else
                mem[mem[PC+1+SEXT(PCoffset9)]]=SR;
```

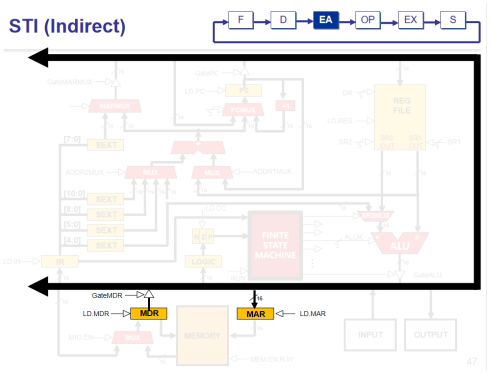
4. 描述:

   由 PC 增量和 SEXT(PCoffset9) 相加得到地址以该地址的内容作为地址, 若地址是特权内存且 PSR[15]=1, 则启动 ACV 异常, 否则把 SR 存入该地址

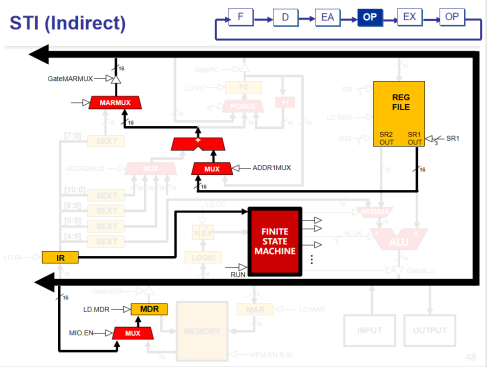
5. 数据通路:



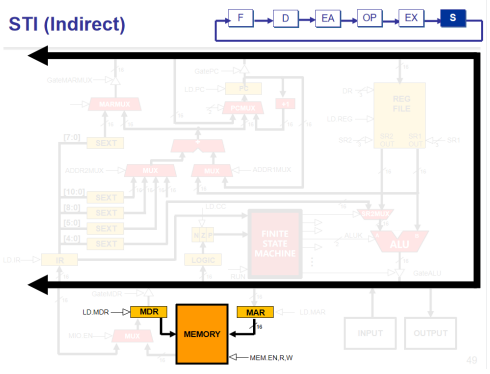
(a) 数据通路-STI 译码和地址计算



(b) 数据通路-STI 取操作数后再次计算地址



(c) 数据通路-STI 再取操作数



(d) 数据通路-STI 存结果

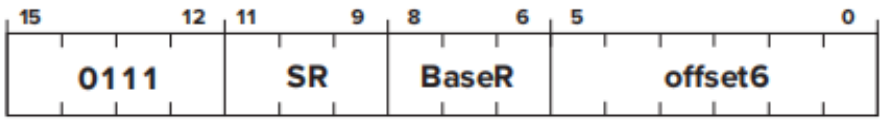
图 17: 数据通路-STI

4.2.14 STR(opcode=0111)

1. 汇编语言:

   STR SR, BaseR, offset6

2. 编码:



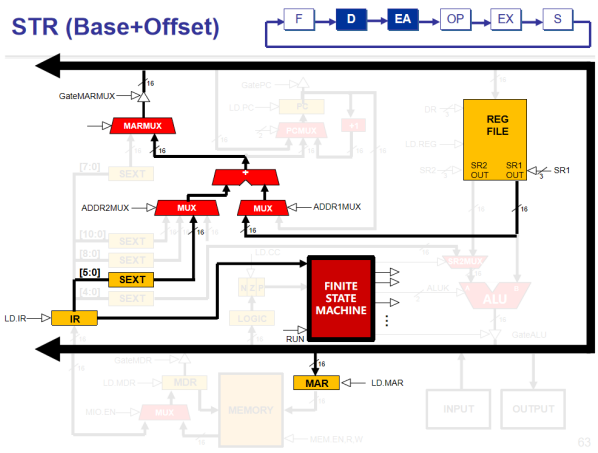
3. 操作:

```
            if ((Computed address is in privileged
memory)&&(PSR[15]==1))
                Initiate ACV exception;
            else
                mem[BaseR+SEXT(PCoffset6)]=SR;
```

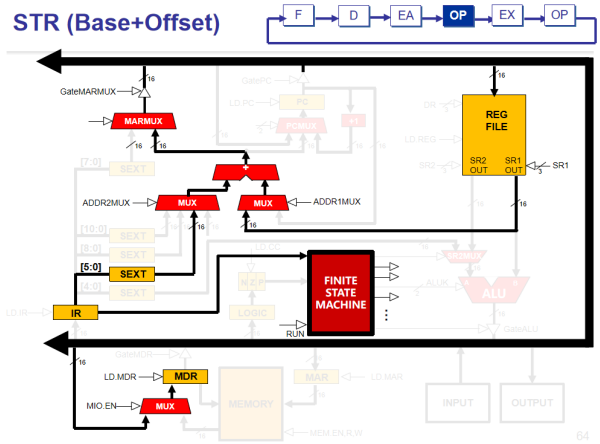
4. 描述:

   由 BaseR 的内容和 SEXT(offset6) 相加得到地址, 若地址是特权内存且 PSR[15]=1, 则启动 ACV 异常, 否则把 SR 存入该地址

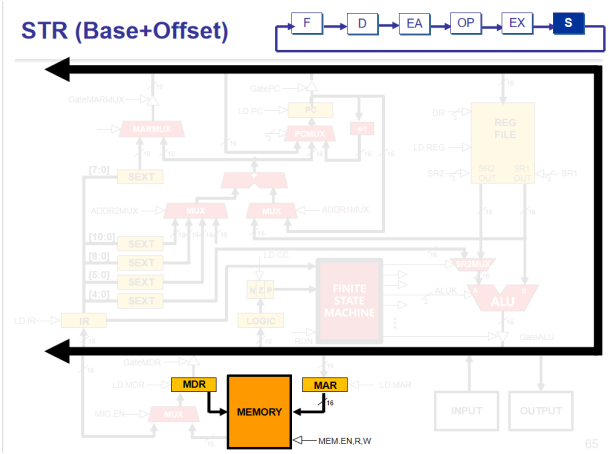
5. 数据通路:



(a) 数据通路-STR 译码和地址计算



(b) 数据通路-STR 取操作数



(c) 数据通路-STR 存结果

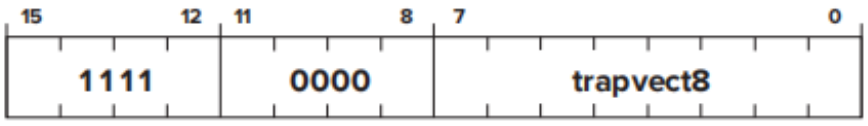
图 18: 数据通路-STR

4.2.15 TRAP(opcode=1111)

1. 汇编语言:

   TRAP trapvector8

2. 编码:



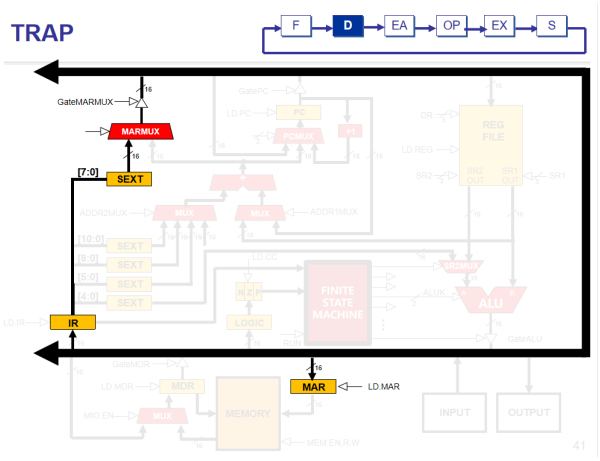
3. 操作:

```
TEMP=PSR;
if(PSR[15] == 1)
{
    Saved USP=R6;
    R6=Saved SSP;
    PSR[15]=0;
}
Push TEMP,PC+1 on the system stack
PC=mem[ZEXT(trapvect8)];
```

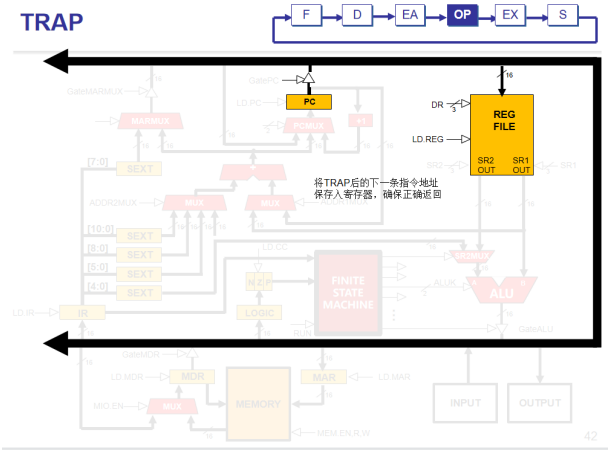
4. 描述:

   如果程序在用户模式下执行, 则必须保存用户堆栈指针并加载系统堆栈指针. 然后将 PSR 和 PC 推送到系统堆栈上.(这使得在服务例程(RTI) 中的最后一条指令完成执行之后, 能够在物理上返回到原始程序中的 TRAP 指令之后的指令.) 然后用 trapvector8 指定的系统调用的起始地址加载 PC. 起始地址包含在存储器位置中, 其地址通过将 trapvector8 扩展到 16 位而获得

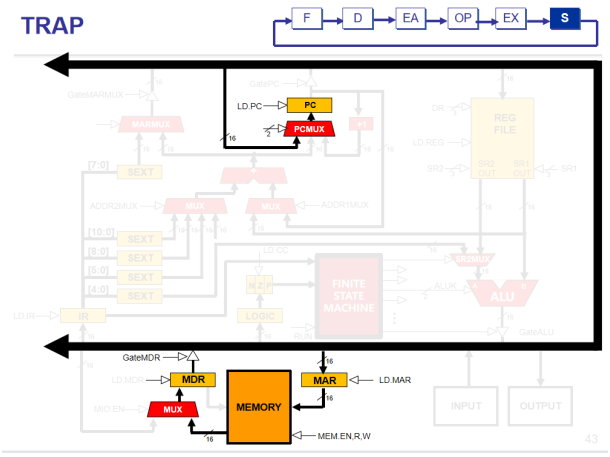
5. 数据通路:



(a) 数据通路-TRAP 译码和地址计算



(b) 数据通路-TRAP 取操作数



(c) 数据通路-TRAP 存结果

图 19: 数据通路-TRAP

6. TRAP 机制

(1) 服务程序: 由操作系统提供, 但以用户身份执行. 这些服务程序是操作系统的组成部分, 起始于各自固定的内存地址,LC-3 最多支持 256 个服务程序.

(2) 起始地址表: 也成为" 系统控制块" 和" 陷入矢量表", 位于内存地址 $x0000 \sim x00FF$, 下面是一个快照:

符号	实际操作	解释
GETC	TRAP x20	阻塞程序并等待键盘输入一个字符（无回显）到R0寄存器 （存ASCII码）
OUT	TRAP x21	输出一个R0中的ASCII码代表的字符
PUTS	TRAP x22	输出R0中地址为起始的字符串，直到遇见\0
IN	TRAP x23	输出提升，阻塞程序并等待键盘输入一个字符（有回显）到R0寄存器 （存ASCII码）
HALT	TRAP x25	终止程序

图 20: 陷入矢量表

(3)TRAP 指令: 如果希望操作系统以用户身份执行某个服务程序, 则使用 TRAP 指令

(4) 链接: 通过链接回到用户程序

7. TRAP 执行

(1) 把 8-bit 的陷入矢量零扩展为 16-bit 地址, 并装入 MAR

(2) 根据 MAR 的内容读出 MDR

(3) 将当前的 PC 值存入 R7, 以实现链接的返回机制

(4) 将 MDR 的内容装入 PC, 至此完成 TRAP 指令

4.3 数据通路的基本部件

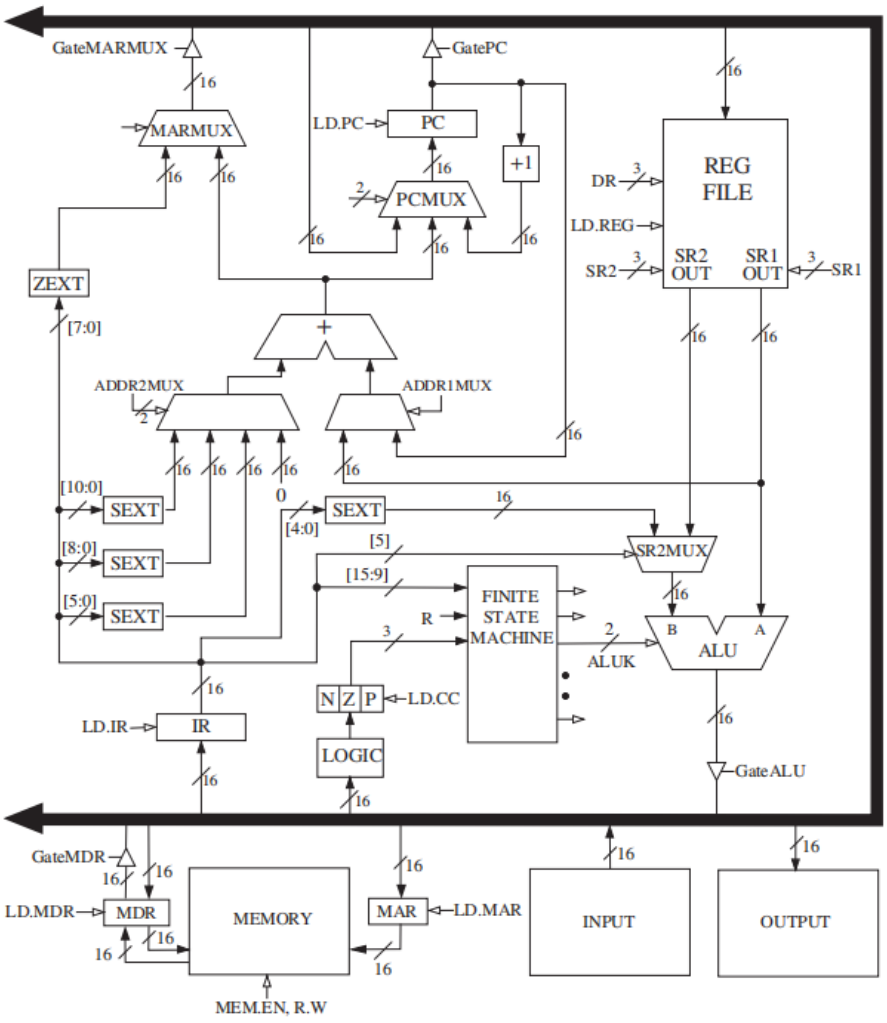


图 21: LC-3 的数据通路

1. 全局总线

   两端都是箭头的黑粗结构是全局总线, 包含 16 根线, 即每次只能传递 16bit 的信息 (总线上同一时刻只能有一个发送者), 每个结构向总线传输数据时有个三角形 (三态门), 以保证只有一个发送者. 接收信号的设备受控于使能信号

2. 内存

   同” 冯 · 诺依曼模型”

3. ALU 和寄存器文件

   ALU 是处理元件. 它有两个输入, 源 1 来自寄存器, 源 2 来自寄存器或指令提供的立即数符号扩展值. 寄存器 (R0 至 R7) 可以提供两个值: 源 1, 由 3-bit 寄存器编号 SR1 控制; 源 2, 由 3-bit 寄存器编号 SR2 控制.SR1 和 SR2 是 LC-3 操作指令中的字段. 第二个寄存器操作数或立即数的符号扩展值的选择由 LC-3 指令的 bit[5] 决定. 注意为 ALU 提供源 2 的多路复用器。该多路复用器的选择线是 LC-3 操作指令的 bit[5].

   ALU 运算的结果是 (a) 存储在其中一个寄存器中的结果, 以及 (b) 三个单位条件码. 注意,ALU 可以向总线提供 16bit, 然后可以将该值写入由 3-bit 寄存器编号号 DR 指定的寄存器中. 此外注意, 提供给总线的 16bit 也被输入到确定该 16bit 值是负、零还是正的逻辑. 相应地设置三个条件码 N、Z 和 P

4. PC 和 PCMUX

   在每个指令周期开始时,PC 通过全局总线向 MAR 提供要获取的指令的地址. 此外,PC 本身的输入由三选一的 PCMUX 提供. 在指令周期的 FETCH 阶段,PC 递增并写入 PC。这显示为 PCMUX 的最右侧输入.

   如果当前指令是控制指令, 则 PCMUX 取决于当前正在处理的控制指令. 如果当前指令是一个条件分支, 并且采取了该分支, 则 PC 加载递增的 PC+PC 偏移量 (通过符号扩展 IR[8:0] 获得的 16 位值). 注意, 这种加法发生在特殊加法器中, 而不是 ALU 中. 加法器的输出是 PCMUX 的中间输入.PCMUX 的第三个输入来自全局总线

5. MARMUX

   MARMUX 控制在执行加载、存储或 TRAP 指令期间, 两个源中的哪个将向 MAR 提供适当的地址.MARMUX 的右输入是通过将递增的 PC 或基址寄存器加至零或 IR 提供的文字值来获得的.PC 或基极寄存器以及文字值取决于正在处理的操作码. 控制信号 ADDR1MUX 指定 PC 或基寄存器. 控制信号 ADDR2MUX 指定要添加四个值中的哪一

个.MARMUX 的左侧输入提供了零扩展 trapvector, 它是调用服务调用所需的

5 汇编语言

标号 (LABEL): 标识一个地址
注释: 写在”;” 之后, 被编译器忽略
可执行映像: 计算机程序的执行实体

5.1 伪操作

1. .ORIG

   标明程序开始的位置 (如.ORIG x3000 表示程序从 x3000 开始)

2. .FILL

   占据一个位置并填充初始值 (如.FILL x86 表示在该位置填充初始值 x86)

3. .BLKW

   占据一串连续的地址 (如.BLKW 5 表示占据 5 个连续的地址)

4. .STRINGZ

   占据一串连续的地址并填充字符串, 在末尾加上空字符 (如.STRINGZ ”Hello,world”, 会把字符以 ASCII 码形式存入一串连续的地址, 并在末尾加上空字符)

5. .END

   表示程序结束了, 停止汇编

6. .EXTERNAL

   我们将一个 LABEL 指定为.EXTERNAL, 则 LC-3 汇编程序将能够为 LABEL 创建符号表条目, 而不是为其分配地址, 而是将符号标记为属于另一个模块. 在链接时, 当所有模块被合并时, 链接器 (管理 “合并” 过程的程序”) 将使用另一个模块中 LABEL 的符号表条目来完成我们修改的翻译.

   通过这种方式, .EXTERNAL 伪操作允许一个模块引用另一个模块中的符号位置. 正确的翻译由链接器解析.

5.2 汇编过程

1. 第一遍扫描: 创建符号表

   丢弃注释, 地址跟踪计数器 LC 的初值由.ORIG 指定, 每识别出一条指令,LC+1, 若指令行头部存在标号, 则创建符号表 [标号, 地址值], 直到.END

2. 第二遍扫描: 生成机器语言程序

   丢弃注释, 初始化地址跟踪计数器计数器 LC, 然后根据第一遍创建的符号表把汇编程序翻译成机器语言

6 数据结构

6.1 子程序

1. 调用/返回机制

   先保存返回调用程序的链接地址, 然后程序跳入子程序, 其中 PC 装入子程序的起始地址,R7 装入返回调用者的链接地址, 子程序的最后一条指令的 RET, 即返回链接地址

2. 寄存器内容的保存和恢复

   分为调用者保存 (caller-save) 和被调用者保存 (callee-save),” 谁知道谁负责”

6.2 栈

定义: 只能在表的一端进行插入和删除的线性表

在内存中实现栈: 由一段连续内存空间和一个寄存器 (栈指针) 组成, 其中寄存器保存栈顶地址, 每个被压入栈中的元素都在内存中占据一个独立的位置, 但在插入和删除时, 其它元素无需移动

1. 压入 (PUSH)

   将 R0 的内容压入栈:

   PUSH ADD R6, R6, # − 1

   STR R0, R6, #0

2. 弹出 (POP)

   将栈顶元素弹出, 并装入 R0:

   POP LDR R0, R6, #0

   ADD R6, R6, #1

3. 下溢出

   对一个空栈执行弹出操作, 会发生” 下溢出” 现象. 因此 POP 程序需检测是否发生下溢出, 若是, 则返回调用程序

4. 上溢出

   对一个满栈执行压入操作, 会发生” 上溢出” 现象. 因此 PUSH 程序需检测是否发生上溢出, 若是, 则返回调用程序

6.3 递归

递归是一种调用自身的函数, 在递归函数本身不是很复杂时考虑使用, 否则时间复杂度很高

6.4 队列

定义: 只能在表的一端插入, 另一端删除的线性表

在内存中实现队列: 由一段连续内存空间和两个寄存器 (队头指针和队尾指针) 组成

如果为队列分配了 n 个内存空间, 当队列元素为 n-1 个时, 队列已满

当头指针 = 尾指针时, 队列为空

R3 作为队头指针,R4 作为队尾指针, 下面是一个头出尾进队列

1. 删除 (Delete)

   删除队头元素, 并装入 R0:

   DELETE ADD R3, R3, #1

   LDR R0, R3, #0

2. 插入 (Insert)

   把 R0 的内容插入队尾:

   INSERT ADD R4, R4, #1

   STR R0, R4, #0

3. 上溢出

   队列已满时插入元素发生上溢出

4. 下溢出

   队列为空时删除元素发生下溢出

7 I/O

7.1 I/O 基本概念

7.1.1 特权、优先级和内存地址空间

1. 特权

   特权是做某事的权利, 例如执行特定的指令或访问特定存储器位置. 并非所有计算机程序有权执行所有指令. 我们将每个计算机程序指定为特权或非特权. 我们经常用主管特权来表示特权. 我们说一个程序是在 Supervisor 模式下执行的, 表示特权, 或者在 User 模式下执行, 表示非特权. 如果程序在 Supervisor 模式下执行, 它可以执行所有指令并访问所有内存. 如果程序在用户模式下执行, 则无法执行. 如果在用户模式下执行的程序试图执行指令或访问需要处于管理员模式的内存位置, 计算机将不允许执行.

2. 优先级

   优先级与执行程序的紧迫性有关. 每个程序都被分配了一个优先级, 指定了与所有其他程序相比的紧迫性. 这允许更紧急的程序中断更不紧急的程序.

3. 处理器状态寄存器 (PSR)

   bit[15] 指定权限, 其中 PSR[15]=0 表示主管权限,PSR[15]=1 表示无权限.bit[10:8] 指定程序的优先级 (PL). 最高优先级为 7(PL7), 最低优先级为 PL0.PSR 还包含条件码的当前值

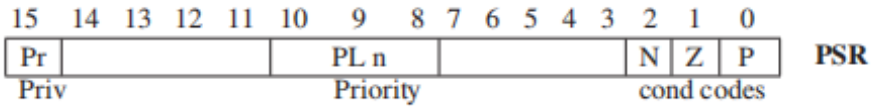


图 22: 处理器状态寄存器 PSR

7.1.2 设备寄存器

任何输入/输出设备至少包含两个寄存器: 一个用来保存跟计算机之间传输的数据, 另一个用来指示当前设备的状态信息

7.1.3 内存映射 I/O 和专用 I/O 指令

指令访问 I/O 设备寄存器时, 需要指明目标寄存器. 通常有两种方法: 一种是使用专门的 I/O 指令访问; 另一种是采用内存操作指令来完成 I/O 操作, 即" 内存映射".

LC-3 采用内存映射 I/O 方式, 内存 $xFE00 \sim xFFFF$ 由外部设备使用

7.1.4 异步 I/O 与同步 I/O

由于 I/O 设备和处理器的工作节奏不一致, 所以说 I/O 设备与 CPU 之间是异步的. 在异步通信方式下, 需要通过一定的协议或握手机制来完成发送和接收.

通过标志来实现同步: 键盘状态寄存器中的标志位. 打字员每键入一个字符, 该 Ready 位就被置位, 每次处理器读取后, 都将该 Ready 位清零. 当处理器检测到 Ready 位时, 使用 load 指令读入键盘寄存器的 ASCII 码值.

7.1.5 中断驱动和轮询

中断驱动 I/O: 由外部设备告诉处理器" 可以读取了"

轮询: 由处理器不断检测外部设备的 Ready 位

7.2 键盘输入

7.2.1 基本输入寄存器

键盘输入寄存器:KBDR 位于 $xFE02$, 其中 bit[7:0] 用于储存输入的 ASCII 码

键盘状态寄存器:KBSR 位于 $xFE00$, 其中 bit[15] 是 Ready 位

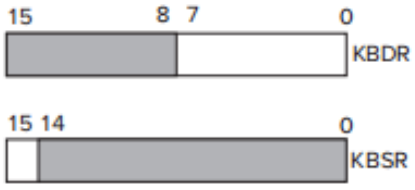


图 23: 键盘设备寄存器

7.2.2 基本输入服务程序

KBSR[15] 用于输入设备和处理器的同步, 当有输入时,KBSR[15] 被置1, 当 LC-3 读取 KBDR[7:0] 后,KBSR[15] 被清零, 此时可以继续输入.

轮询方式下,KBSR[15] 不断被检测. 同时,KBSR 的同步机制保证了输入会且仅会被读取一次

下面的程序读入 KBDR 的 ASCII 码值存入 R0, 并跳转

```
STRAT LDI R1, KBSR
BRzp STRAT
LDI R0, KBDR
BRnzp NEXTTASK
KBSR .FILL xFE00
KBDR .FILL xFE02
```

7.2.3 内存映射输入的实现

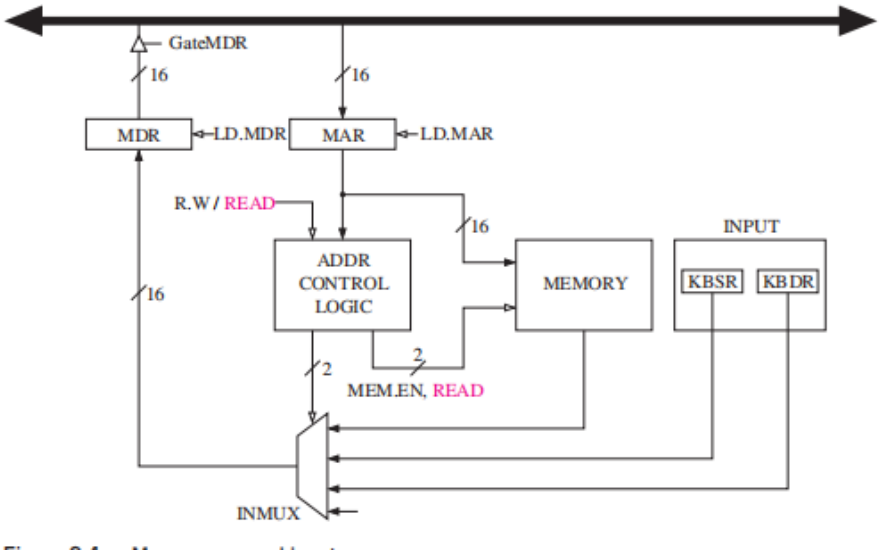


图 24: 数据通路-内存映射输入

7.3 显示器输出

7.3.1 基本输出寄存器

显示器输出寄存器:DDR 位于 $xFE06$, 其中 bit[7:0] 用于储存输出的 ASCII 码

显示器状态寄存器:DSR 位于 $xFE04$, 其中 bit[15] 是 Ready 位

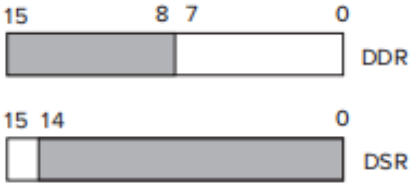


图 25: 显示器设备寄存器

7.3.2 基本输出服务程序

DSR[15] 用于输入设备和处理器的同步, 当有输出时,DSR[15] 被清零, 表示” 忙”, 当 LC-3 输出 DDR[7:0] 后,DSR[15] 置 1, 此时可以继续输出.

轮询方式下,DSR[15] 不断被检测. 同时,KBSR 的同步机制保证了输入会且仅会被读取一次

下面的程序读入 R0 中的字符并输出到 DDR, 并跳转

```
STRAT LDI R1, DSR
BRzp STRAT
STI R0, DDR
BRnzp NEXTTASK
DSR .FILL xFE04
DDR .FILL xFE06
```

7.3.3 内存映射输出的实现

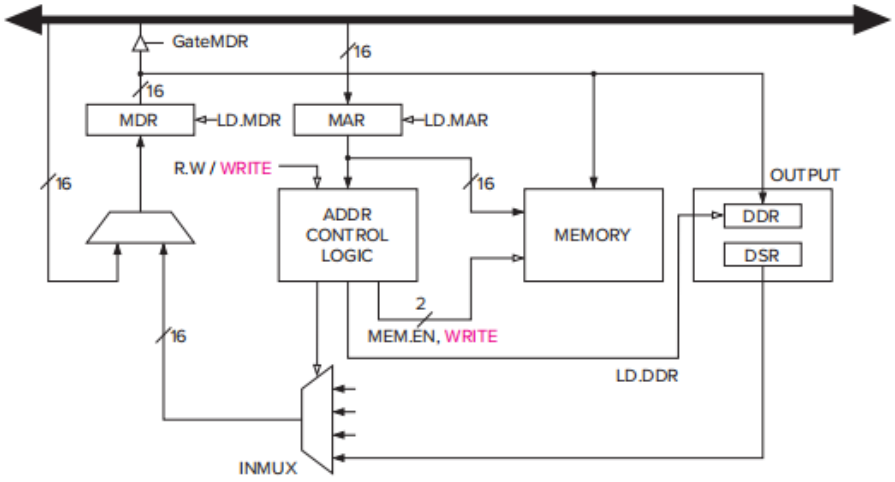


图 26: 数据通路-内存映射输出

总结: 内存映射 I/O 的数据通路:

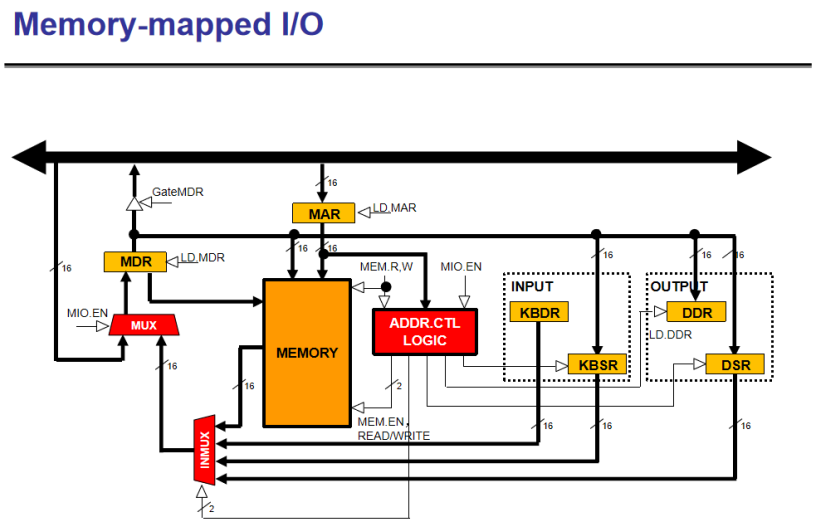


图 27: 数据通路-内存映射 I/O

7.4 中断驱动 I/O

7.4.1 什么是中断驱动 I/O

中断驱动 I/O 的本质是 I/O 设备能实现以下功能:
(1) 强行终止当前程序的运行;
(2) 使得处理器处理 I/O 设备请求;
(3) 恢复被中断程序的执行, 让它感觉好像什么都没发生过一样;
中断驱动 I/O 包括两部分内容:
(1) 产生中断信号, 终止当前执行程序;
(2) 处理该中断请求;

7.4.2 Part1: 中断信号的产生

设备是否必须, 以及是否能够中断处理器, 必须具备下面几个条件:
(1)I/O 设备自身确实需要服务;
(2) 设备有请求服务的权限;
(3) 设备中断请求的优先级高于当前处理器正在运行程序的优先级;

1. 来自设备的中断信号 对于条件 (1), 即对应设备的 Ready 位. 对于条件 (2),KBSR 和 DSR 有" 中断使能标志位"bit[14], 若为 1, 则表示允许发送中断信号. 若 (bit[15] AND bit[14])=1, 则发出中断信号.

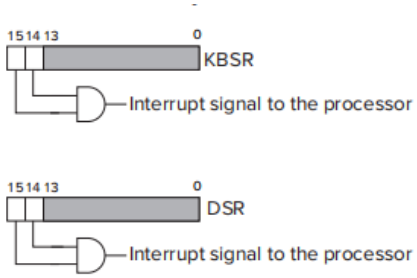


图 28: 中断使能位

2. 中断优先级 从优先级编码器中选出的优先级如果比当前程序的优先级高, 则产生 INT 信号.

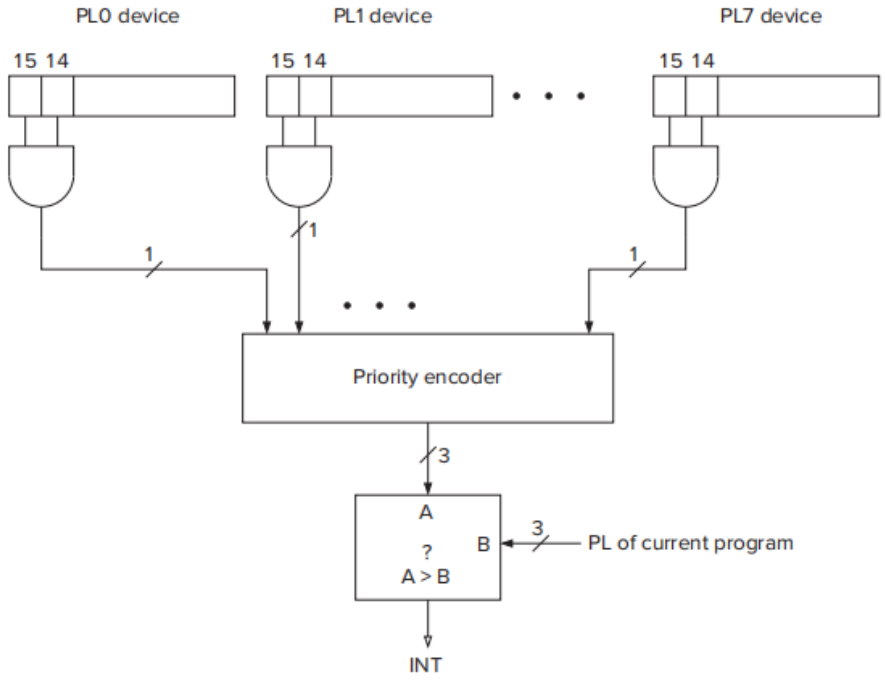


图 29: INT 信号的产生

3. 中断检测 在程序周期的最后一步 (存结果) 中增加对 INT 信号的检测, 若检测有效, 则在返回下一个取指令节拍之前, 控制单元将完成两项工作:

(1) 保存足够的状态信息, 以便之后正确恢复被中断程序的执行环境;

(2) 将即将服务于该设备请求的程序入口地址装入 PC;

7.4.3 Part2: 处理中断请求

处理中断请求包括三个阶段:

(1) 启动;

(2) 执行;

(3) 返回;

1. 启动

1. 保存被中断程序的状态 程序的状态是程序所有资源内容的快照. 它包括作为程序一部分的存储器位置的内容和所有通用寄存器的内容. 它还包括 PC 和 PSR.

启动中断的第一步是保存足够的正在运行的程序状态, 以便在 I/O 设备请求得到满足后, 程序可以继续运行。这意味着, 在 LC-3 的情况下, 保存 PC 和 PSR.

我们假设通用寄存器的内容由服务程序保存 (callee),LC-3 只保存 PC 和 PSR

在中断服务例程开始前, 若被中断的程序处于用户模式, 则用户栈指针 (USP) 被保存到 Saved USP, 然后从 Saved SSP 把超级用户栈指针 (SSP) 装入 R6, 最后把 PSR 和 PC 压入超级用户栈.

2. 加载中断服务例程的状态 发出中断时, 设备向处理器发送一个 8-bit 的矢量值 (类似 TRAP), 优先级最高的矢量值被选中 (称为 INTV), 随后扩展为一个 16-bit 的地址, 即中断矢量表 ($x0100 \sim x01FF$) 的一个表项地址. 中断矢量表中每一项都是一个中断服务程序的起始地址. 随后被选中地址对应的内容被装入 PC.

对于 PSR, 在中断开始时,PSR[2:0] 被初始化为 0,PSR[15] 被置 0,PSR[10:8] 即为中断请求者的优先级别

2. 执行 由于 PC 包含中断服务例程的起始地址, 服务例程将执行,I/O 设备的要求将得到服务.

3. 返回 每个中断服务例程中的最后一条指令是 RTI(即 Return from trap or interrupt). 当处理器最终访问 RTI 指令时,I/O 设备的所有要求都得到了满足.

中断服务例程的 RTI 指令的执行简单地包括将 PC 和 PSR 从超级用户栈中弹出, 并将它们恢复到处理器中的正确位置. 条件码现在恢复到程序中断时的状态, 以防中断程序中的后续 BR 指令需要它们.PSR[15] 和 PSR[10:8] 现在反映了即将恢复的程序的特权级别和优先级. 如果被中断程序的特权级别是非特权的, 则必须调整栈指针, 即保存的超级用户栈指针 (SSP) 和加载到 R6 中的用户栈指针 (USP). 如果程序没有中断,PC 将恢复到下一个执行的指令的地址.

在中断发生之前, 所有这些事情都是一样的, 程序可以恢复, 就像什么都没有发生一样.

8 附录

8.1 完整的 LC-3 数据通路

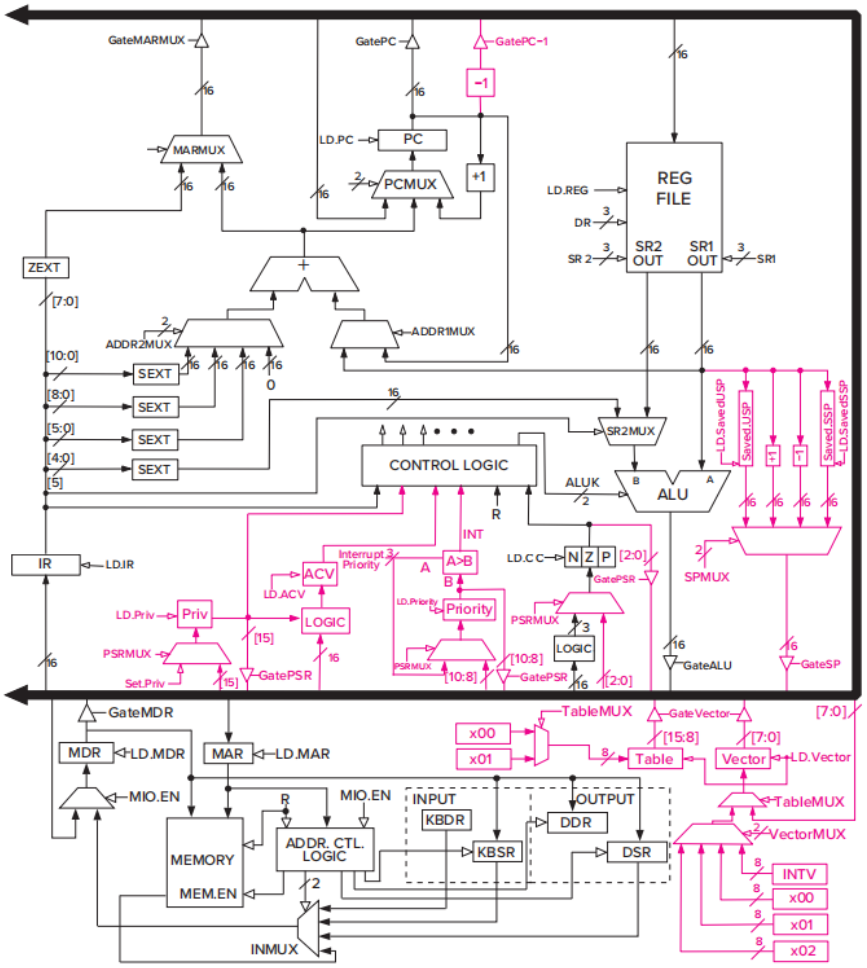


图 30: 完整的 LC-3 数据通路

8.2 LC-3 状态机

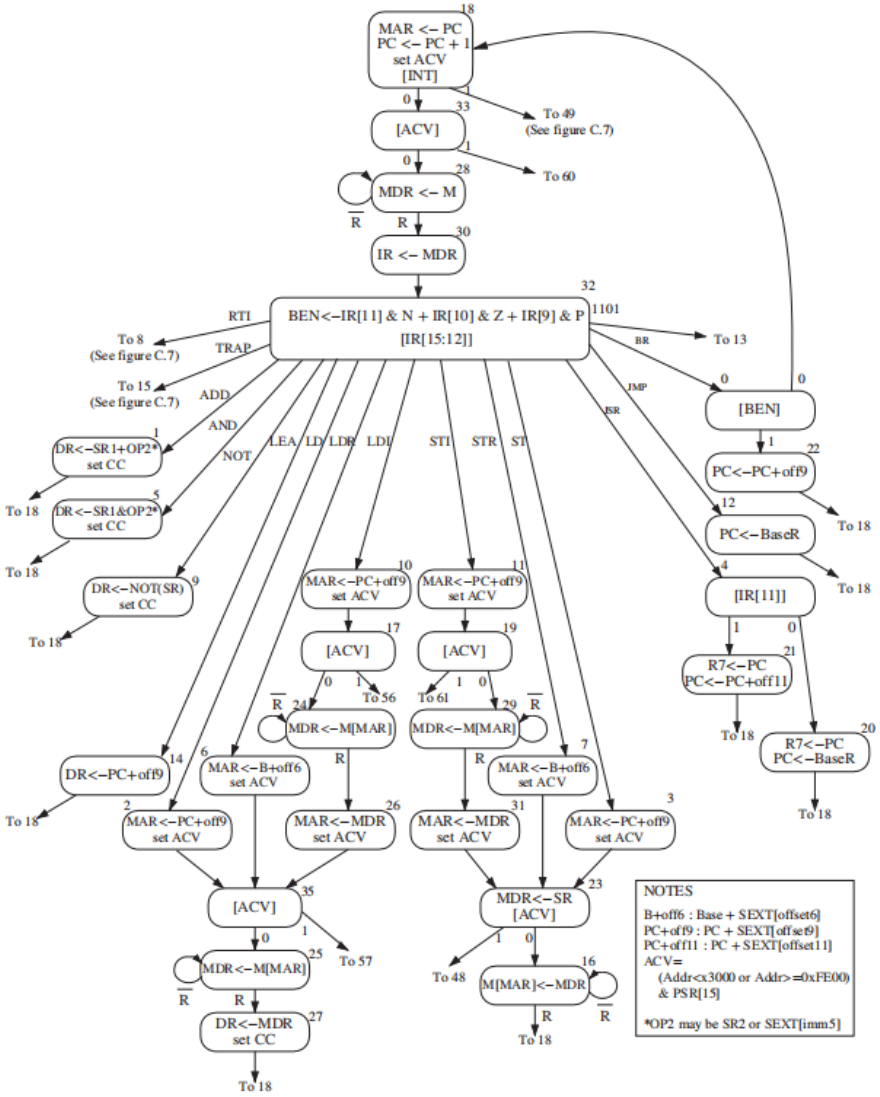


图 31: LC-3 状态机

8.3 LC-3 中断控制状态机

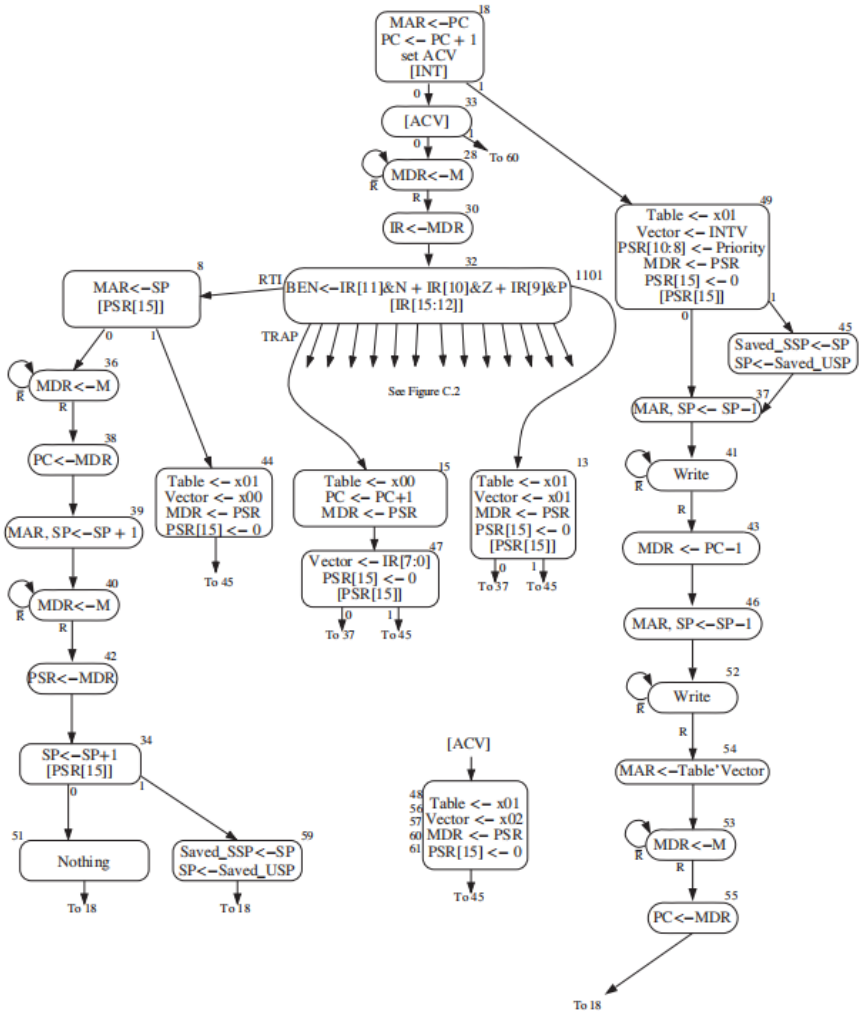


图 32: LC-3 中断控制状态机