计算机系统概论

> 更改：
>
> LEA不再设置条件码
>
> TRAP不再以R7作为返回链接，而采用压栈和出栈

第二章 bit，数据类型及运算

> bit是信息的基本单位能表示一个0或1。

无符号整数与有符号整数。

补码：(2's complement)

> 负数数值位取反加一，ps：对整位取反加一得其相反数

进制转化

算术运算

- 加法直接加，减法补码运算，乘除使用移位操作。
- **符号位扩展**，正数补0，负数补1（以补码表示）位数不同的二进制运算先扩展为同位。**十进制运算大法**
- 溢出，同位相加可能会溢出，向前进位与符号位不同代表溢出产生错误

逻辑运算

- 与或非，异或同或，以及按位进行上述操作。
- 位矢量：以n bit代表n个单元，0，1表示其工作与否。

浮点数 P26 (Floating-point converse to FIxed-point)

通常采用IEEE标准，共三十二位。第一位为符号位(sign)，后八位表示指数（1<=n<=254)(有效计为n-127)，在后23位表示小数。

> 需要注意可能的溢出问题

ASCII码：last page

第三章 数字逻辑

MOS晶体管

- NMOS：高电平导通
- PMOS：低电平导通

由MOS管构成逻辑门 P35

AND OR NOT is complement

要证明其他门或门组合完备，需要证明它能实现与，或，非。

摩根定律：用于转换非

组合逻辑电路

- 编码器
- 译码器（Decoder)
- 复用器（选择器)(MUX)
- 全加器（FULL Adder）(减法器对另一输入取非)
- Incrementer(增量器)
- 可编程逻辑阵列（PLD）

储存单元（Basic Storage Elements)

Combinational 组合的
Sequential 时序的 锁存器 寄存器 触发器

内存 P46

内存大小通常为寻址空间*寻址能力

- 寻址空间：n bit 表示其有2^n个内存单元（n 通常也代表地址线根数）
- 寻址能力：m bit 表示其有每个内存单元有 m bit 大小

A向量表示地址，WE 为write enable.

状态机（State Machine)

有限状态机（Finite State Machine）：每个状态都是系统处于一个状态的快照，包含：

- 有限系统状态
- 有限输入输出
- 状态迁移方向
- 状态转换原因

在计算机中状态的改变通常由时钟信号进行引导。

数据通路（Data Path)

- Control Unit
- Processing Unit
- Memory Unit
- I/O

4.冯.诺伊曼模型

LC3 Data Path

- Memory: Storage of information (data/program)
- Processing Unit: Computation/Processing of Information
- Input: Means of getting information into the computer. e.g. keyboard, mouse
- Output: Means of getting information out of the computer. e.g. printer, monitor
- Control Unit: Makes sure that all the other parts perform their tasks correctly and at the correct time.

Memory

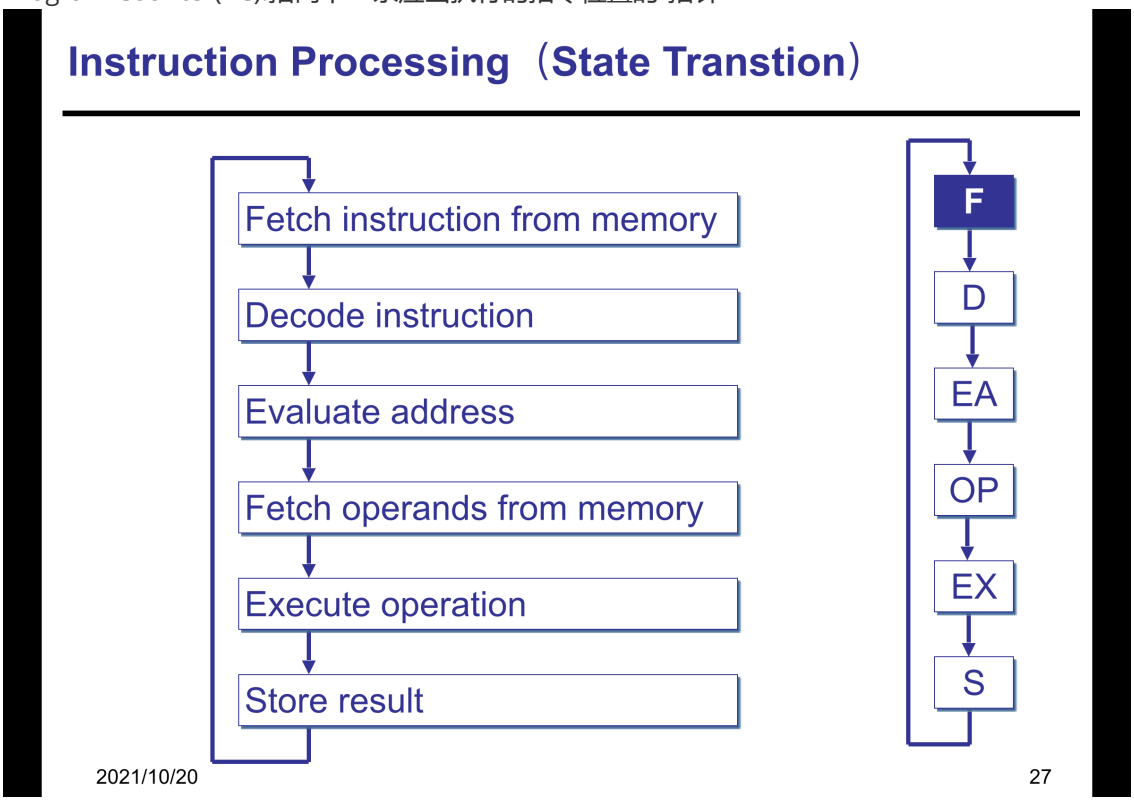
- Address | Contents
- Basic Operation: LOAD | STORE
- Interface to memory: (内存接口)
  - MAD: Memory Address Register
  - MDR: Memory Data Register

Processing Unit

- Function Unit
- Register
- Word Size（常见的32位处理器，64位处理器。字长与处理能力有关）

Control Unit

- Instruction Register(IR):存储当前指令
- Program Counter(PC):指向下一条应当执行的指令位置的"指针"



Instruction

- opcode:操作指令
- operands:操作数或地址
- Instruction Set Architecture(ISA):指令集结构

5.The LC3 Operate Instructions

Memory of LC3:

寻址空间：2^{16}(或65536)

寻址能力(字大小)：16bit

Registers:

- eight general-purpose register(八个通用寄存器)：$R_0 - R_7$
- other register

opcode

- 15 opcodes（1101 保留没有定义暂不可使用）
- Operate opcode: ADD,AND,NOT
- Data Movement: LD,LDI,LDR,LEA,ST,STR,STI
- Control instruction: BR, JSR/JSRR, JMP(RET), RTI, TRAP

TIP: the opcodes with underline will set condition codes: **N: negative Z: zero P: positive**
Attention:LEA no longer sets the condition codes.

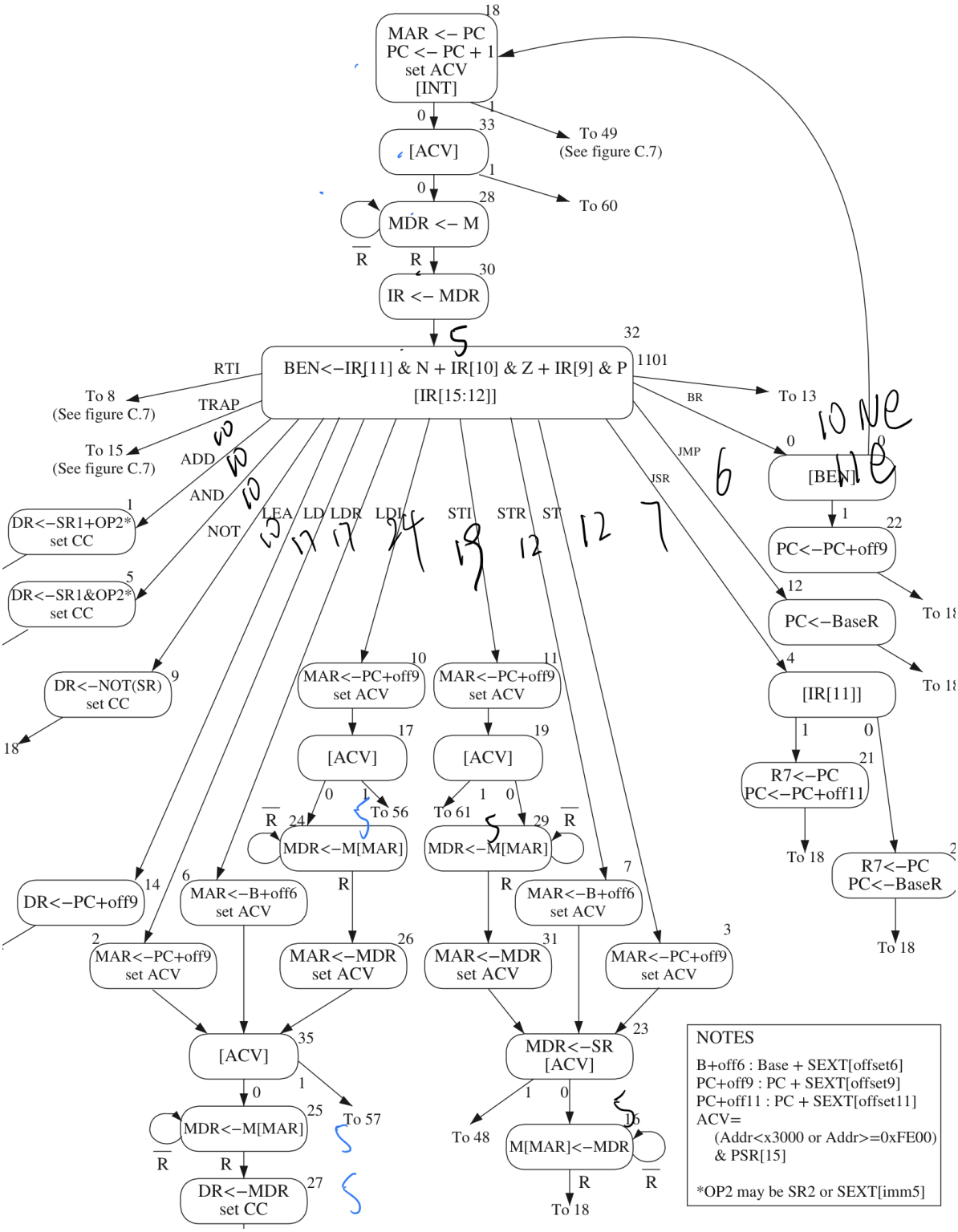
TRAP x23使用R0暂存读入值。与返回相关的指令**不会使用**R7暂存部分恢复信息。

Addressing Modes

- 非寻址：立即数，通用寄存器
- 寻址：PC相关寻址，间接寻址，base+offset寻址

Data Path

- 全局总线
- 内存
- ALU和寄存器
- PC和PCMUX
- MAR和MARMUX



6.编程

问题求解

系统分解

- 算法：有限性，确定性，可计算性（可有计算机完成）。
- 结构化编程与系统分解：将问题分解为独立的小问题，使得每个问题都能够被程序员独立完成并测试。

三种结构

顺序，条件与循环（BR指令即可完成条件与循环）

程序错误的定位与解决：调试

7.汇编语言

> 机械语言太过繁琐，指令过于难记，因此我们使用自然语言直接指代机械语言中的指令，同时使用标志代替位置跳转时所需的计算，这无疑方便了许多。

汇编语言的指令

指令格式

> LABEL OPCODE　OPERAND ;COMMENT、

LABEL指代该指令的位置，通常在第一遍编译时确定。

OPCODE一对一指代LC3的机器码指令只是一种自然语言的替代。

OPERAND可以是通用寄存器也可以是立即数，与机器码不同的是立即数可用LABEL替代省去计算的不便。

伪操作（.____）

.ORIG _____告诉编译器接下来一段代码放在内存的哪个位置。

.FILL　告诉汇编器占用该空间放入指定值

.BLKW 告诉汇编器占用指定大小空间，常用于申请内存。

.STRING 告诉汇编器占用指定字符串长度的空间，并依次填入字符（会额外填入0x0000）。

.END 告诉汇编器有效代码结束，.END后的代码会被汇编器忽略。

具体的指令示例

参见课本342面附录A，这里省略。

汇编的过程

> 由于LABEL的存在，在一开始程序并不知道LABEL指什么，直到读到它。如果它出现使用它的后面汇编将难以进行。

第一次扫描

创建符号表将每个LABEL与其对应的内存地址记录下来。

第二遍扫描

按照符号表，和汇编指令和机器码指令的对应关系将汇编指令逐句翻译为LC3机器码指令。

Extra knowledge:

可执行映像

多目标文件

8.输入与输出

输入与输出的概念

设备寄存器

- 系统中存在专门存放IO设备信息的设备寄存器，一个IO设备通常存在多个设备寄存器与之对应存储设备状态，以及近期的IO操作，以及与计算机的数据传输。

内存映射与专用IO指令

前者将设备寄存器当作特殊一点的寄存器使用一般的内存操作指令访问数据（被通常使用），这些特殊的地址位通常会被保留如在LC3中0xFE00-0xFFFF保留给外部设备。

后者定义专门的指令用于访问设备寄存器（通常很少）。

同步IO与异步IO

CPU工作的频率是十分快的，而IO通常是相对缓慢且难以预知的。也就是说**IO操作一般来说是异步的**，这时的通信需要靠协议来确保工作的稳定可靠性。

使用ready位是一种最简单的协议，当数据准备就绪时，ready置1，cpu获取信息后将ready置0 。

不过IO的频率是恒定已知的那么我们可以预置时间让cpu取数据，而无需ready位，这样的**IO是同步的。**

中断驱动（interrupt）与轮询（pulling）

中断驱动：由设备主导，当其有待执行请求时可能会试图中断cpu正在执行的任务（通常不应当中断一条指令的执行）让cpu将原任务挂起优先响应其请求，对于IO操作这表现**为当设备数据就绪时，主动通知cpu。**

轮询：由cpu主导，通常对于紧急的数据，cpu可能会亲自**反复"询问"设备数据是否已就绪。**

来自键盘的输入

键盘的设备寄存器

KBDR(键盘数据寄存器)(FE02),KBSR(键盘状态寄存器)(FE00)

KBDR[7:0]有效，KBSR[15]有效

基本输入服务程序

当键盘输入一个字符到KBDR[7:0]时，自动将KBSR[15]置1，当LC3程序取走数据时自动将KBSR[15]置0 。

每个字符有且仅有一次被读取的机会。

来自显示器的输出

显示器的设备寄存器

DDR(数据寄存器)(FE06),DSR(状态寄存器)(FE04)

DDR[7:0]有效，DSR[15]有效

基本输出服务程序

当LC3输入一个数据到DDR时自动清零DSR表示显示器正在处理该字符的输出，当输出完成时自动将DSR置1表 示已经就绪。

中断驱动

中断信号的产生

- 中断的过程
  - 产生中断信号，中止当前程序。
  - 处理中断请求，返回被中断程序。
- 中断的条件
  - 设备确实需要服务。
  - 设备有请求的权限。
  - 设备请求的优先级高于正在被服务的程序的优先级。
- 来自设备的中断请求

  ready位也即前面提到的KBSR,DSR。
- 中断优先级

  LC3定义了八个优先级PL0-8，多个设备可能有多个请求，通过优先编码器选择其中最为紧迫的请求，最后通过比较器与当前程序优先级进行比较，判断是否要进行中断，产生**INI信号**
- 中断的检测

  指令周期在store后会对INI信号进行检测，如果有效，则会保存足够的信息后fetch中断请求的指令。

中断的启动与执行

- 被中断程序的状态保存，这包括PC，PSR。其中PSR包含特权级别，运行优先级，和条件码。

  > 通常情况下中断IO不必考虑保存通用寄存器，这默认服务程序在使用他们时会保存，在完成时恢复。
  >
  > PC,PSR会被压入supervisor stack中。
- 中断服务程序状态装入，这通常包括根据矢量扩展而来的PC，PSR。此时PSR[15]为0处于特权模式下，PSR[10:8]设为请求设备的优先级，PSR[2:0]全置为0。
- 中断服务 完成任务

中断返回

中断程序最后是RTI指令，会将特权栈顶的PSR,PC弹出栈分别装入PC，和PSR。至此中断完成，控制权交回用户程序手中。

内存映射在LC3中的数据通路

9.TRAP程序和子程序

TRAP程序

trap机制

- 服务程序集合--由系统提供
- 起始地址表：包含256个服务程序的起始地址存放在0x0000-0x00FF中。（表A-2）
- TRAP指令，用户调用时相当于以用户程序身份执行系统服务程序。
- 链接：服务程序完成时返回用户程序的机制。

trap的工作

- 扩展8-bit矢量装入MAR中
- 根据MAR读取内容到MDR
- 保存PCsuperuser stack
- 将MDR装入PC开始服务程

寄存器内容的保存与恢复

调用其他程序包括系统程序会有可能改变寄存器原有的值，这要求我们要保护好有用的值，一般遵循谁知道谁保存的原则。

启动与执行

- 保存PC，PSR

  > PSR[15]为特权状态，PSR[10:8]为运行优先级，PSR[2:0]为条件码NZP

  将PC，PSR压入superuser Stack
- 装入中断程序的状态包括起始地址，PSR
- 中断返回弹出PSR，PC恢复原程序执行。

子程序

JSR/JSRR和RET

栈

抽象数据类型（ADT）

后进先出的数据结构

操作（通常R6会被作为栈指针）

压入：注意上溢出。

```
PUSH ADD R6 R6 #-1
     STR R0 R6 #0
```

```
PUSH AND R5 R5 #0
     LD R1 MAX
     ADD R2 R6 R1
     BRZ Failure
     ADD R6 R6 #-1
     STR R0 R6 #0
     RET
Failure ADD R5 R5 #1
     RET
MAX  .FILL XC005
```

弹出：注意下溢出。

```
POP  LDR R0 R6 #0
     ADD R6 R6 #-1
```

```
POP  AND R5 R5 #0
     LD R1 MIN
     ADD R2 R6 R1
     BRZ Failure
     LDR R0 R6 #0
     ADD R6 R6 #1
     RET
Failure ADD R5 R5 #1
     RET
MIN  .FILL XCOOO
```

队列

先进先出的数据类型

操作（通常R3会被用作队头指针，R4会被用作队尾指针）

入队出队

出队

```
        LD R2, LAST
        ADD R2,R3,R2
        BRnp SKIP_1
        LD R3,FIRST
        BR SKIP_2
SKIP_1  ADD R3,R3,#1
SKIP_2  LDR R0,R3,#0 ; R0 gets the front of the queue
        RET
LAST    .FILL x7FFB ; LAST contains the negative of 8005
FIRST   .FILL x8000
```

入队

```
        LD R2, LAST
        ADD R2,R4,R2
        BRnp SKIP_1
        LD R4,FIRST
        BR SKIP_2
SKIP_1  ADD R4,R4,#1
SKIP_2  STR R0,R4,#0 ; R0 gets the front of the queue
RET
LAST    .FILL 7FFB ; LAST contains the negative of 8005
FIRST   .FILL x8000
```

同时需要注意上溢和下溢出

```
            AND R5,R5,#0 ; Initialize R5 to 0
            NOT R2,R3
            ADD R2,R2,#1 ; R2 contains negative of R3
            ADD R2,R2,R4
            BRz UNDERFLOW; code to remove the front of the queue and return
success.
UNDERFLOW   ADD R5,R5,#1
            RET
```

队空时头尾指针指向同一个寄存器，队满时尾指针为头指针位置减一。