「计算机系统概论」TA Session 5

2022-2023 短学期计算系统概论课程

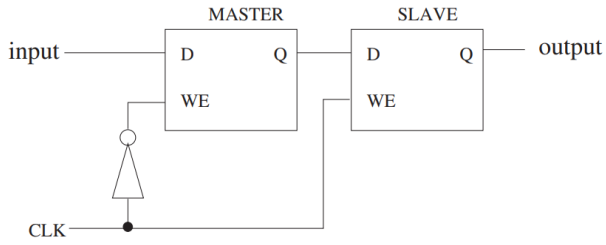
By @HobbitQia

Homework 3

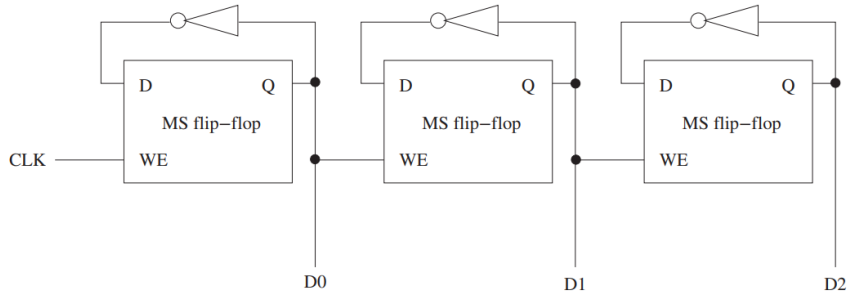
Flip-flop

3.53

★3.53 The master/slave flip-flop we introduced in the chapter is shown below. Note that the input value is visible at the output after the clock transitions from 0 to 1.



Shown below is a circuit constructed with three of these flip-flops.



Fill in the entries for D2, D1, D0 for each of clock cycles shown

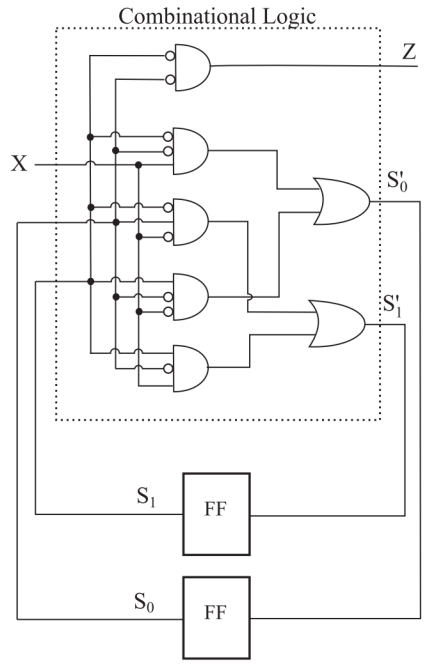
	cycle 0	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6	cycle 7
D2	0							
D1	0							
D0	0							

In ten words or less, what is this circuit doing?

FSM

3.61 画出下面电路图的有限状态机

The logic diagram shown below is a finite state machine.



真值表如下：

S1	S0	X	Z	S1'	S0'
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	0	0

FSM (Cont.)

实际上，FSM 也分为两种模型

- Moore 模型：输出仅与状态有关。
- Mealy 模型：输出同时与状态、输入有关。

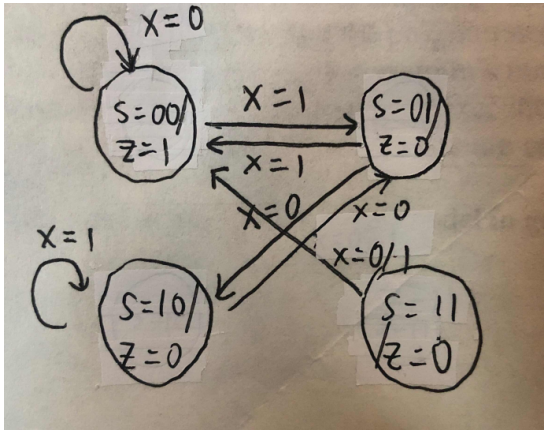
回顾 FSM state diagram 的画法：

- 圆圈表示状态（状态名写在圈内）
- 有向弧表示状态转化（从现态到次态）
- 有向弧上标明外部输入和输出（如果有），圆圈内也可以标明当前状态的输出（如果有）。
  对于 Moore 模型，输出应该写在圈内。对于 Mealy 模型，输出应该写在有向弧上。以 Input/Output 的形式。

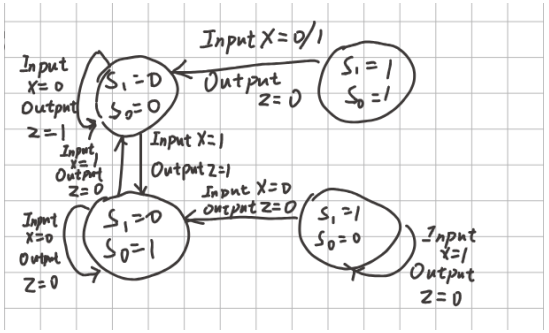
FSM (Cont.)

在 3.61 中，用两种模型都是可以的。但需要注意只有 S1S2 两个存储单元用来保存状态，因此至多只能有四个状态。

Moore 模型：



Mealy 模型：



Homework 4

Homework 4

5.2 内存的寻址能力是 64 位，那么 MAR 和 MDR 是多少位?
地址的位数和内存存储的数据位数没有关系！因此这里无法确定 MAR 的位数。

5.16 以下情形，哪种寻址方式更好?

- You want to load an array of sequential addresses. 意思是你想要加载一系列地址连续的数据，而不是只是加载地址（LEA is wrong.）就像数组，从 a[0], a[1], a[2] 依次访问。

答案: Base+offset mode. 因为 Indirect/PCoffset 在这种情形下很麻烦，而且 PCoffset 有范围限制。对于 Base+offset 来说，我们找到连续地址的起始点后，只需要把基寄存器加一即可继续访问后面的内存。

Chapter 7: Aseembly Language

Aseembly Language: Review

- **high-level**
  High-level languages tend to be **ISA-independent**.
  e.g. C, C++, Java, Python
  把 high-level 语言转化为 ISA（机器码），需要 compiler 编译。
- **low-level**
  Assembly languages are low-level languages, and they are very much **ISA-dependent**. It is usually the case that each ISA has only one assembly language.
  把 low-level 语言转化为 ISA，需要 assembler 汇编。

实际上汇编器只是把指令翻译成对 01 串（具体翻译过程见后面），编译器会对指令做优化。

An Assembly Language Program - Instructions

LC-3 汇编程序里每条指令的格式如下：

```
Label Opcode Operands ; Comment
```

其中；后面为注释，注释和标签为可选的内容。

An Assembly Language Program - Instructions

LC-3 汇编程序里每条指令的格式如下：

Label Opcode Operands ; Comment

其中；后面为注释，注释和标签为可选的内容。

- opcode
  这里的 opcode 并不是 0101 的数字，而是 symbolic name（用来代指 opcode，便于阅读）***e.g.*** ADD, AND, or LDR rather than 0001, 0101 or 0110.
- comments
  注释只是给人看的，不会被汇编器翻译。***e.g.*** LD R0, ASCII ; Load the ASCII template The message "Load the ASCII template" is a comment.

Instructions (Cont.)

- operands
  操作数的数量由指令决定 ***e.g.*** NOT 指令需要两个操作数，ADD 指令需要三个操作数。
  操作数可以是寄存器，立即数，标签。
  立即数需要标明进制 *# for decimal, x for hexadecimal, and b for binary*. 需要注意的是教材上说这里的 *#b 是必须写的，但是实际上 LC-3 模拟器会默认为十进制，以教材为准。
- labels
  label 标签是用来标识**内存地址**的，相当于地址的一个别名。
  在 LC-3 中，标签可以是 1~20 个字母数字，开头必须是字母（除了保留字，比如不能和指令同名）
  常用于 load/store 类指令 和 BR/jmp 跳转指令。 ***e.g.*** BRnzp LOOP, LD R1, TEMP

Pseudo-Ops (Assembler Directives)

Pseudo-Ops 伪指令也被称为 Assembler Directives 汇编指令。
并不是在执行阶段执行，而是在汇编阶段就已经执行了。
伪指令的格式和汇编指令一样：Label Pseudo-op Operand ; Comment，因此也可以在前面写标签。

- .ORIG
  .ORIG 告诉汇编器接下来的指令的起始地址。***e.g.*** 我们通常会在开头写 .ORIG x3000，这样我们的程序的第一条指令就会被放在 x3000
- .FILL
  .FILL 会把 operand 的值填充到当前地址（.FILL 的地址）。

Pseudo-Ops (Cont.)

```
1       .ORIG x3000
2       LD R2, LABEL
3       HALT
4   LABEL .FILL x1000
5       .END
```

左边的代码中，.FILL 的地址为 x3002，通过 .FILL 指令我们设置了 MEM[x3002]=x1000。这样我们就可以通过 LD 指令来取出这个 x1000 这个值。

- BLKW (a **BL**oc**K** of **W**ords)
  预留出 operand 个字的空间。***e.g.*** 假设当前地址为 x3000 那么 .BLKW 10 就会将从 x3000 开始的是个地址全部留出来（从 x3000 到 x3009），下一条指令的地址为 x300A。
  教材上并没有说对于留出的空间会做什么操作，但是实际上 LC-3 模拟器会将这些空间全部初始化为 0。

Pseudo-Ops (Cont.)

- .STRINGZ
  .STRINGZ 后面的操作数是引号包围的字符串，指令留出从当前地址开始的 n+1 个位置（n 是字符串的长度），并将字符串的每个字符依次填充到地址中去（一个地址放一个 ASCII 字符），最后一个位置放 x0000（表示字符串终止）。

e.g. 假设当前地址为 x3000 那么 .STRINGZ "Hello" 就会将从 x3000 开始的 6 个地址全部留出来（从 x3000 到 x3005），下一条指令的地址为 x3006。其中

```
x3000: x0048
x3001: x0065
x3002: x006C
x3003: x006C
x3004: x006F
x3005: x0000
```

Pseudo-Ops (Cont.)

- .END
  .END 用来告诉汇编器当前程序达到末尾了，后面的部分不需要再翻译。 需要注意的是:
  - .END 并不会结束程序的执行，只是一个分隔符。
  - 实际上一个程序内可以有多个 .END。每一组 .ORIG .END 表示程序的某一段。

The Assembly Process

assemble soutiona1.asm outfile 可以把 asm 文件汇编成 obj 文件，这个过程称为 assembly process，这其中需要 2 complete passes 两次完整的扫描。

- The First Pass: Creating the Symbol Table
  symbol table 符号表是一个映射，将 label 映射到对应的地址。
  第一次扫描我们仅找到所有的 label，以及其对应的地址。
- The Second Pass: Generating the Machine Language Program
  翻译 LC-3，如果遇到 label 就把刚刚符号表中的地址替代 label 进行计算。如果地址超过了能访问的范围（比如 LD 了一个距离 x300 的地址），那么汇编器会报错。

Beyond the Assembly of a Single Assembly Language Program

计算机执行程序时，这个实体称为可执行映像。可执行映像通常由多个不同的程序员独立创建的模块创建（也是不同的目标文件）。

如果我们在程序里使用了不存在的 label（这个 label 实际上应该存在于另一个模块）我们可以用 .EXTERNAL label 来声明。这样在链接的时候，所有的模块被合在一起，我们就能找到这个标签的位置。

更详细的细节，可以参考 *Chapter 7 Linking* of the book *Computer Systems: A Programmer's Perspective*.

谢谢大家

Question?