中国科学技术大学
University of Science and Technology of China

计算机组成原理
--数字电路背景知识

卢建良 lujl@ustc.edu.cn

2022年春季学期

提纲

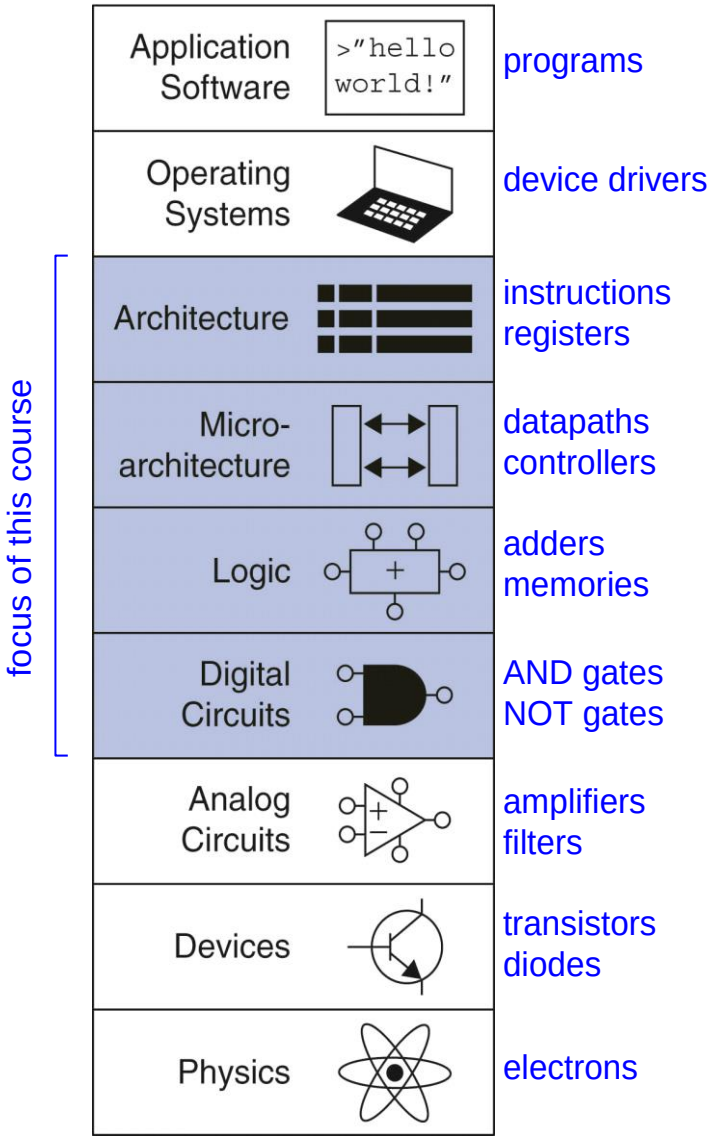
- 1. 引言
- 2. 门、真值表、逻辑方程
- 3. 组合逻辑电路
  - 译码器、多选器、两级逻辑（PLA）、ROM、无关项、逻辑单元阵列
- 4. 硬件描述语言
  - 数据类型和操作、Verilog代码结构、复杂组合逻辑的表示
- 5. 构建基本算数逻辑单元（ALU）
  - 1位ALU、64位ALU、修改64位ALU以适应RISC-V、用Verilog定义RISC-V ALU
- 6. 快速加法：超前进位
- 7. 时钟
- 8. 存储元件：触发器、锁存器和寄存器
  - 触发器和锁存器、寄存器堆、使用Verilog描述时序逻辑

提纲-续

- 9. 存储元件：SRAM和DRAM
  - SRAM、DRAM、错误修正
- 10. 有限状态机（FSM）
- 11. 定时方法
  - 电平敏感的时钟控制、异步输入和同步器
- 12. 现场可编程设备
- 13. 本章小结

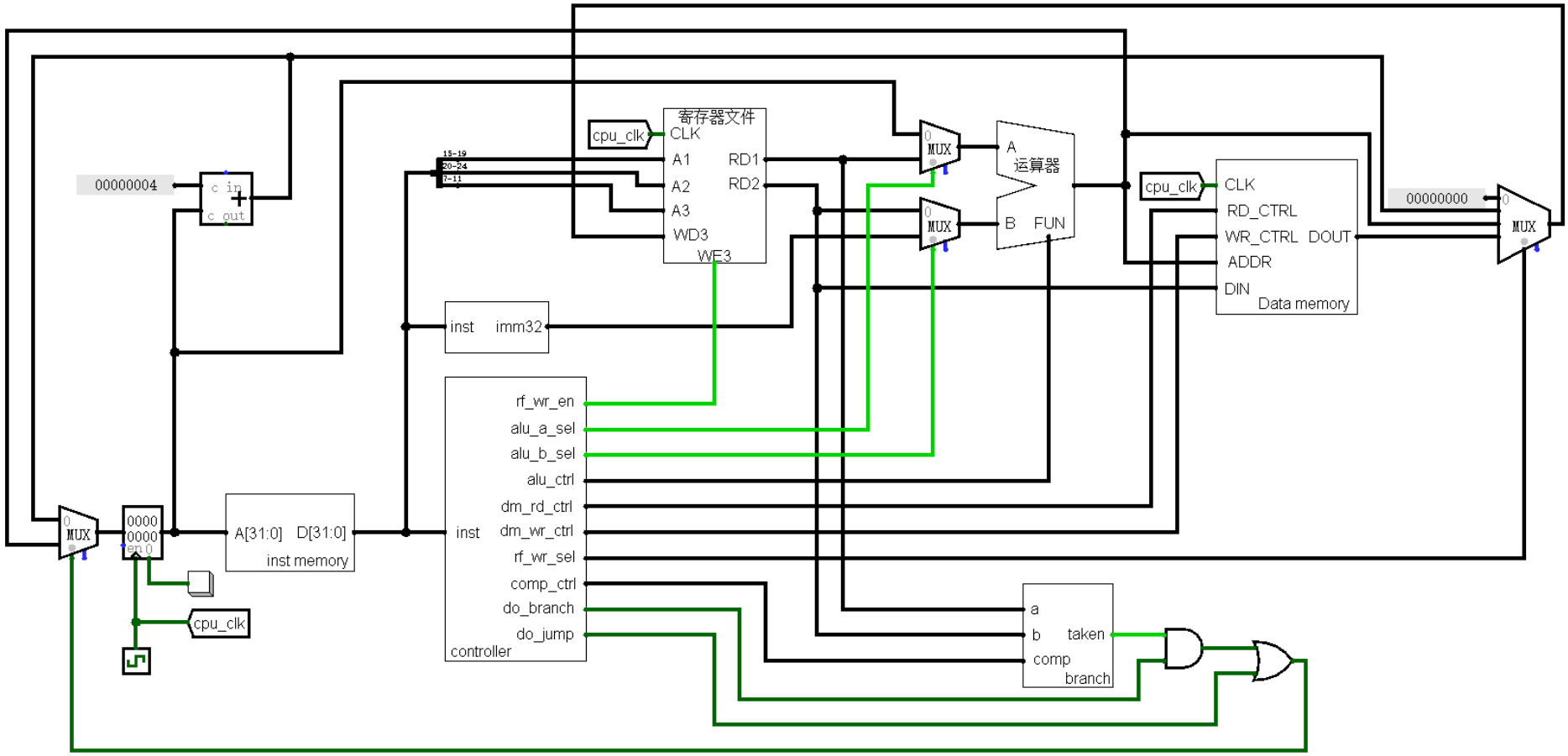
A.1 引言

- 本章简要讨论逻辑设计的基础知识
- 为理解本课程内容提供必要的背景知识储备
- 在适当的地方配有Verilog代码片段
- 完整的Verilog教程网站：http://staff.ustc.edu.cn/~han/CS152CD/Content/Tutorials/Verilog/VOL/main.htm
- Verilog在线测评网站
  - https://hdlbits.01xz.net/wiki/Step_one
  - https://verilogoj.ustc.edu.cn/oj/



A.1 引言

- 复习数字电路知识是为设计RV32I CPU做准备

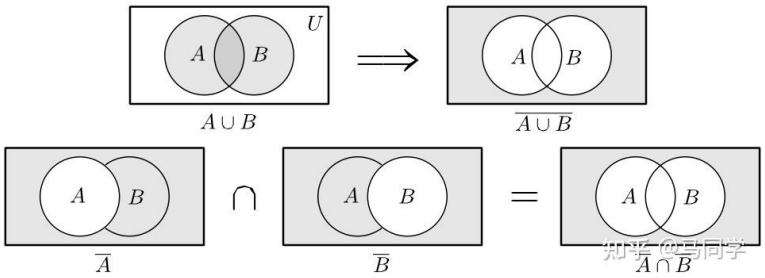


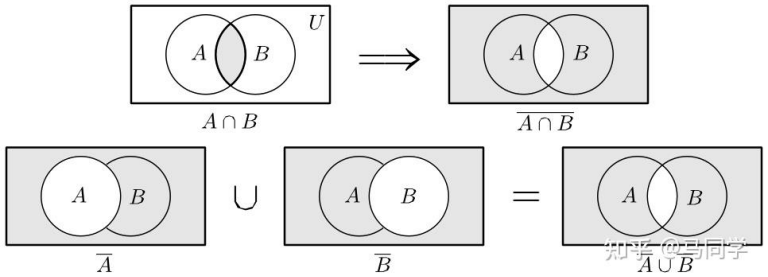
A.2 门、真值表和逻辑方程

- 数字电路是现代计算机的内部核心
- 数字电路有两个稳定的电平状态：高电平 vs 低电平
  - 1 vs 0、 真 vs 假、 有效 vs 无效、正 vs 负、高 vs 低、 阳 vs 阴
  - 隐患：电路处于不稳定的电平状态时会如何?
- 组合逻辑电路 vs 时序逻辑电路
  - 电路中是否含有存储单元
  - 输出是否与之前的状态有关
- 真值表
  - 输入逻辑变量所有取值的组合与其对应的输出逻辑函数值构成的表格
  - 缺点：表项增长太快、不容易理解
  - 改进：有时采用仅列举非零输出表项的简化真值表
- 布尔代数
  - 或操作： A + B
  - 与操作： A · B
  - 非操作： /A

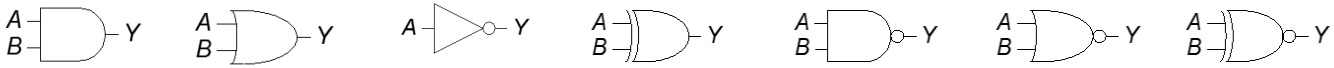
A.2 门、真值表和逻辑方程-续

- 布尔代数-续
  - 布尔代数中的基本定律：恒等定律、0/1定律、互补律、交换律、结合律、分配律
  - 德摩根定律 $\overline{A \cup B} = \overline{A} \cap \overline{B}$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$

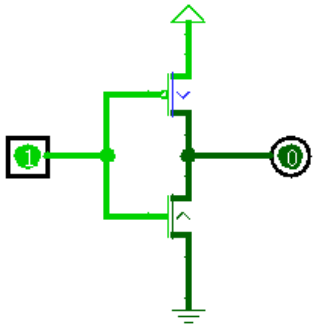




- 门（gate）：实现基本逻辑函数的单元，与、或、非、与非等
  - 逻辑门



  - 逻辑门电路：具有逻辑门功能的电路单元
  - 可以使用CMOS工艺实现逻辑门电路
  - 在Logisim软件中进行行为仿真
  - 万能门电路：或非门、与非门
    - 任何逻辑电路都可以只使用该类型构建

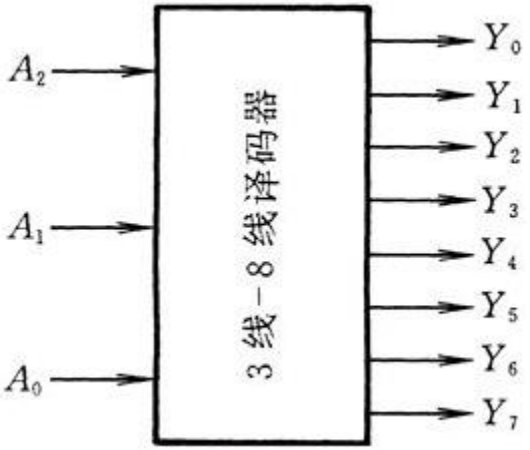


$Y = \overline{A}$

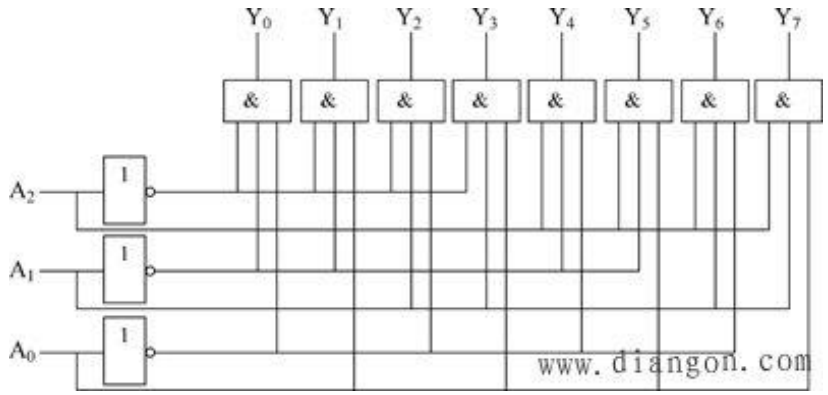
A	Y
0	1
1	0

A.3 组合逻辑电路

- 译码器（decoder）
  - 具有n位输入和2^n个输出的逻辑单元
  - 每种输入组合仅对应一个有效输出
  - 最常见的为3-8译码器
  - 用途：
    - 用来生成不同时使用的多个功能模块的片选或使能信号
    - 用来实现IO接口的扩展，如数码管
  - 与其相对应的是编码器，如何实现？



输入			输出							
A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



A.3 组合逻辑电路

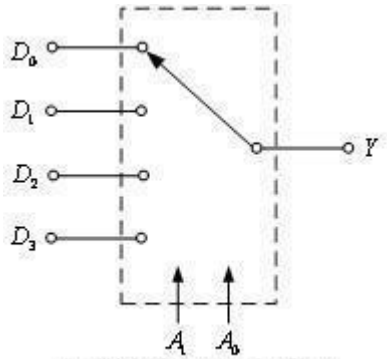
- 练习：
- 用Verilog实现3-8译码器
- 用Verilog实现8-3编码器

A.3 组合逻辑电路

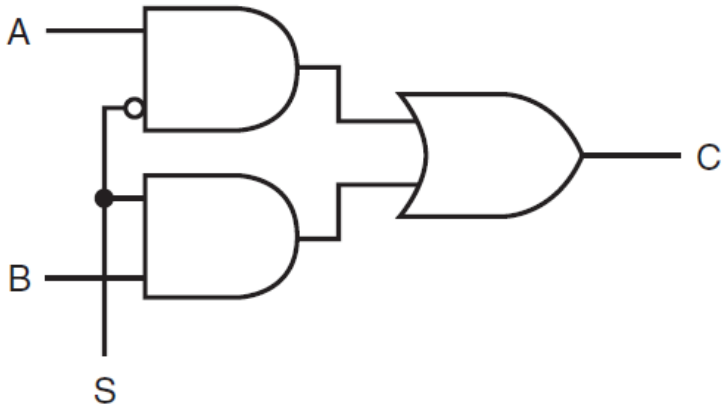
- 多路选择器（Multiplexor/Mux）
  - 多路选择器是数据选择器的别称。在多路数据传送过程中，能够根据需要将其中任意一路选出来的电路，叫做数据选择器，也称多路选择器或多路开关
  - 逻辑函数：

    C = （A · /S） + （B · S）
  - Verilog实现

```
module mux2(A,B,S,C);
    input A,B,S;
    output C;
    assign C = S ? B : A ;
endmodule
```

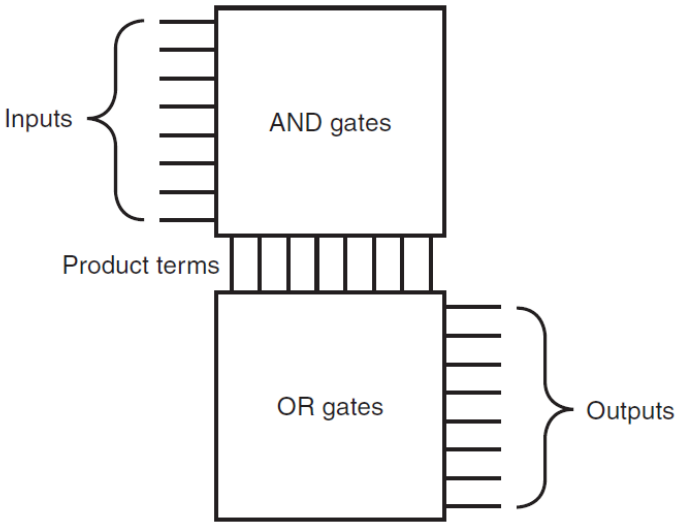


4选1数据选择器原理示意



A.3 组合逻辑电路

- 两级逻辑、PLA
  - 任何逻辑都可以写成“或-与式”或者“与-或式”的表示形式
  - 或-与式（ product of sums ） $E = \overline{(\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{C} + B) \cdot (\overline{B} + C + A)}$
  - 与-或式（sum of products） $E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (B \cdot C \cdot \overline{A})$
  - 与-或式对应于可编程逻辑阵列（PLA）的常用结构化实现
  - PLA： **P**rogrammable **L**ogic **A**rray

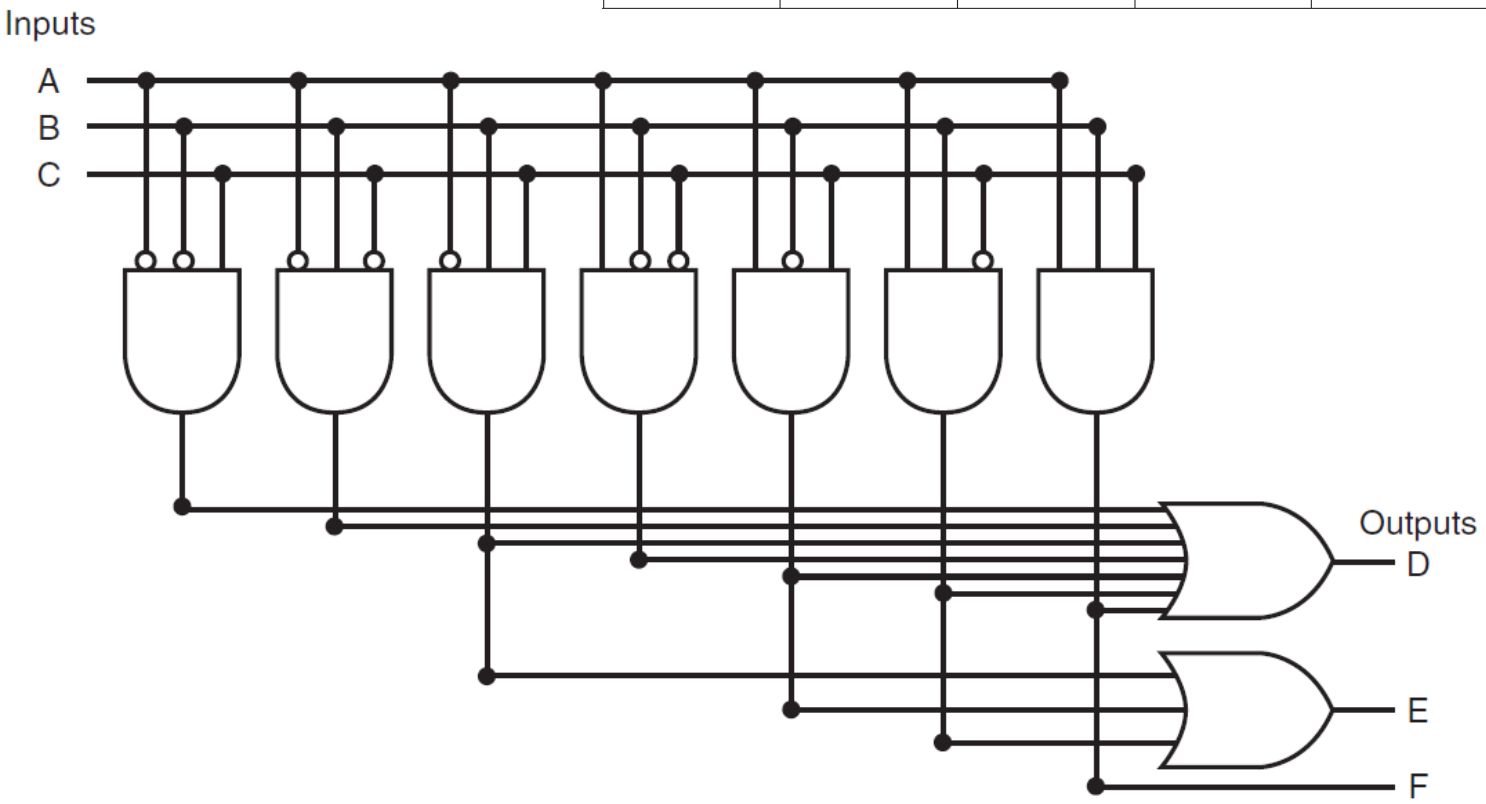


Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

A.3 组合逻辑电路

- 两级逻辑、PLA
  - 例：用PLA实现真值表所描述的电路

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1



A.3 组合逻辑电路

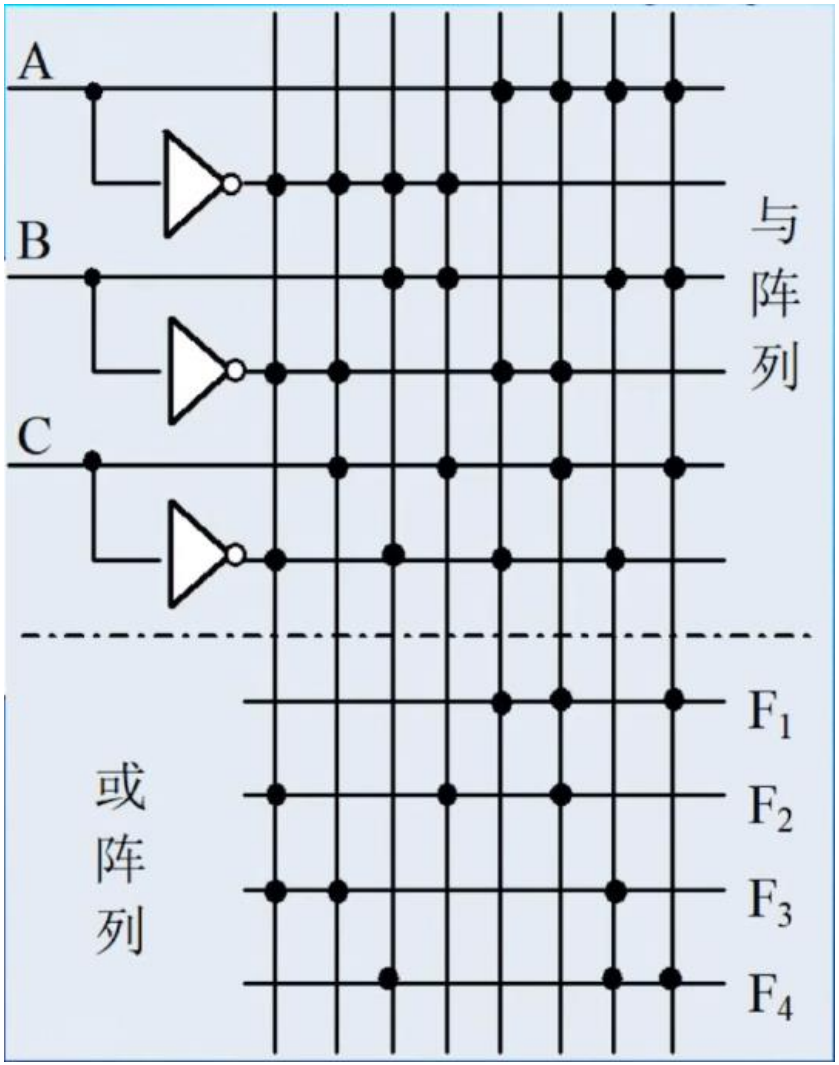
- 两级逻辑、PLA
  - 例：用PLA器件实现下列逻辑函数

$$F_1 = A\bar{B} + AC$$

$$F_2 = \bar{A}BC + A\bar{B}C + \bar{A}\bar{B}\bar{C}$$

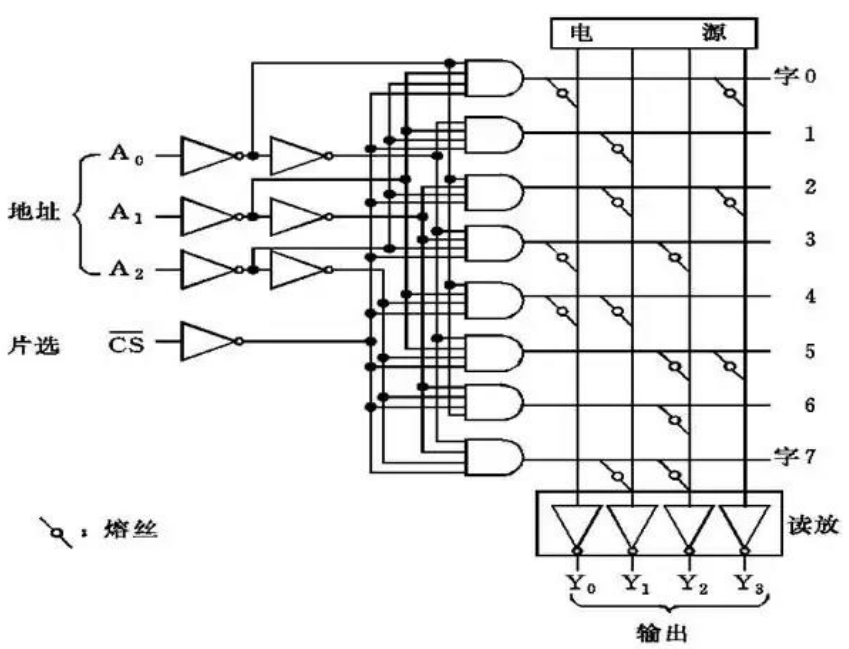
$$F_3 = \bar{A}\bar{B} + AB\bar{C}$$

$$F_4 = \bar{A}B\bar{C} + AB$$

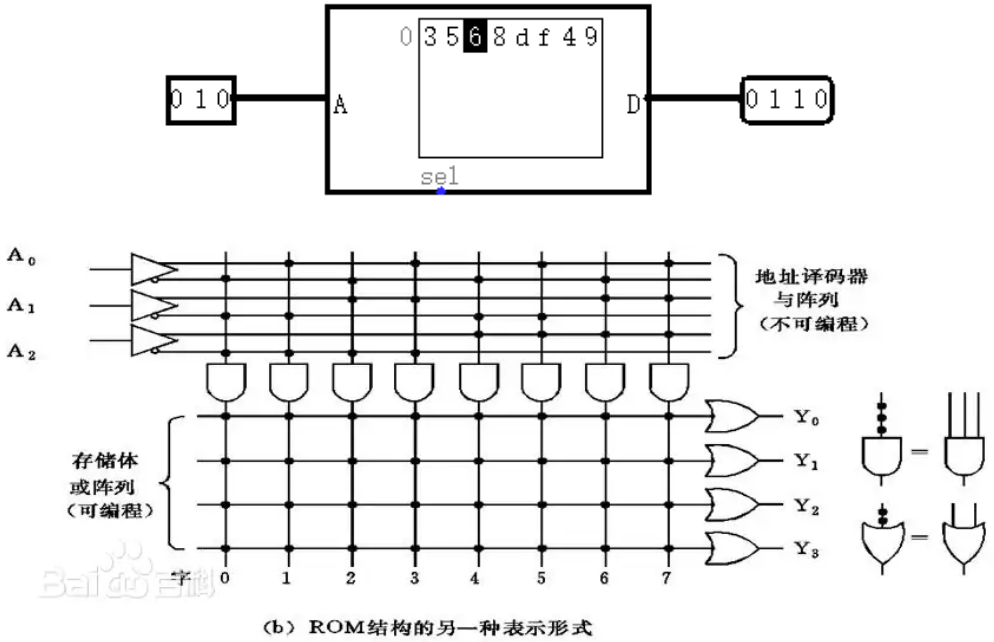


A.3 组合逻辑电路

- ROM
  - Read-Only Memory，只读存储器
  - 一种可以长期保存信息的存储器，具有断电后信息仍可继续保存的特点，在正常工作时只可读取数据，而不能写入数据
  - 是用于实现组合逻辑函数结构化逻辑形式的另一方式



（a）熔丝型8×4ROM原理图



（b）ROM结构的另一种表示形式

A.3 组合逻辑电路

- 无关项
  - 分为输入无关项和输出无关项
  - 无关项对逻辑函数的优化至关重要
  - 例题：

Consider a logic function with inputs A, B, and C defined as follows:

- If A or C is true, then output D is true, whatever the value of B.
- If A or B is true, then output E is true, whatever the value of C.
- Output F is true if exactly one of the inputs is true, although we don't care about the value of F, whenever D and E are both true.

Inputs			Outputs		
A	*B*	*C*	*D*	*E*	*F*
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	0

A.3 组合逻辑电路

- 无关项
  - 输出无关项化简

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	1	1	X
1	0	0	1	1	X
1	0	1	1	1	X
1	1	0	1	1	X
1	1	1	1	1	X

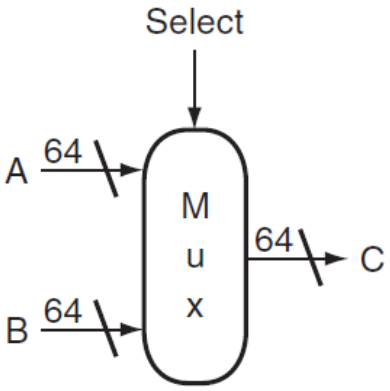
  - 输入无关项化简

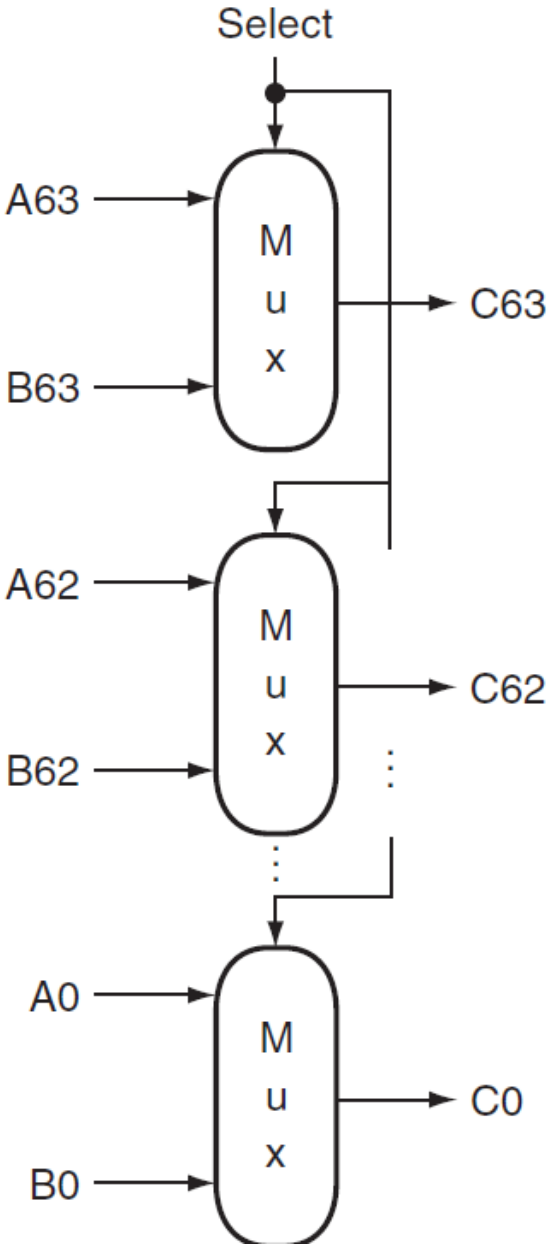
Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	1	1
X	1	1	1	1	X
1	X	X	1	1	X

A.3 组合逻辑电路

- 逻辑单元阵列，总线
  - 数据处理时经常需要对整个数据字（32bit or 64bit）进行处理，因此需要构建逻辑单元阵列，又称总线，bus
  - PS:总线也用于指示具有多信号源和多设备共享的线路集合
  - Verilog实现：

```
module mux64(
input [63:0] A,B,
input S,
output [63:0] C);
    assign C = S ? B : A;
endmodule
```





单选题 1分

设置

Parity is a function in which the output depends on the number of 1s in the input. For an even parity function, the output is 1 if the input has an even number of ones. Suppose a ROM is used to implement an even parity function with a 4-bit input. Which of A, B, C, or D represents the contents of the ROM?

Address	A	B	C	D
0	0	1	0	1
1	0	1	1	0
2	0	1	0	1
3	0	1	1	0
4	0	1	0	1
5	0	1	1	0
6	0	1	0	1
7	0	1	1	0
8	1	0	0	1
9	1	0	1	0
10	1	0	0	1
11	1	0	1	0
12	1	0	0	1
13	1	0	1	0
14	1	0	0	1
15	1	0	1	0

A

B

C

D

提交

A.4 使用硬件描述语言

- 硬件描述语言
- Verilog中的数据类型和操作
  - 两种基本数据类型：wire 、 reg
  - 定义数据向量：reg [31:0] x; wire [63:0] y;
  - 定义数据组：reg [31:0] regfile[0:31];
  - reg、wire信号可能的取值有：0、1、X、Z
  - 常量值可以指定为2,8,10,16进制数
    - `4'b0100` specifies a 4-bit binary constant with the value 4, as does `4'd4`.
    - `-8'h4` specifies an 8-bit constant with the value −4 (in two's complement representation)

Values can also be concatenated by placing them within { } separated by commas. The notation `{x{bitfield}}` replicates `bitfield` `x` times. For example:

- `{32{2'b01}}` creates a 64-bit value with the pattern 0101 … 01.
- `{A[31:16],B[15:0]}` creates a value whose upper 16 bits come from `A` and whose lower 16 bits come from `B`.

A.4 使用硬件描述语言

Check Yourself

Which of the following define exactly the same value?

1. `8'bimoooo`
2. `8'hF0`
3. `8'd240`
4. `{{4{1'b1}},{4{1'b0}}}`
5. `{4'b1,4'b0)`

A.4 使用硬件描述语言

- Verilog程序的结构
  - `initial` constructs, which can initialize `reg` variables
  - Continuous assignments, which define only combinational logic
  - `always` constructs, which can define either sequential or combinational logic
  - Instances of other modules, which are used to implement the module being defined
- Verilog复杂组合逻辑的表示
  - assign

```
module half_adder (A,B,Sum,Carry);
   input A,B; //two 1-bit inputs
   output Sum, Carry; //two 1-bit outputs
   assign Sum = A ^ B; //sum is A xor B
   assign Carry = A & B; //Carry is A and B
endmodule
```

  - always
    - 阻塞赋值：用于组合逻辑，用“=”表示
    - 非阻塞赋值：用于时序逻辑，用“<=”表示

A.4 使用硬件描述语言

- Verilog复杂组合逻辑的表示
  - always

```
module RISCVALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [63:0] A,B;
   output reg [63:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0; goes anywhere
   always @(ALUctl, A, B) //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1:0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0; //default to 0, should not happen;
      endcase
endmodule
```

A.4 使用硬件描述语言

Check Yourself Assuming all values are initially zero, what are the values of A and B after executing this Verilog code inside an `always` block?

```
C = 1;
A <= C;
B = C;
```

阻塞赋值示例：

```
always@(posedge clk)
begin
        a = 1;
        b = a;
        c  = b;
end
```

不建议使用

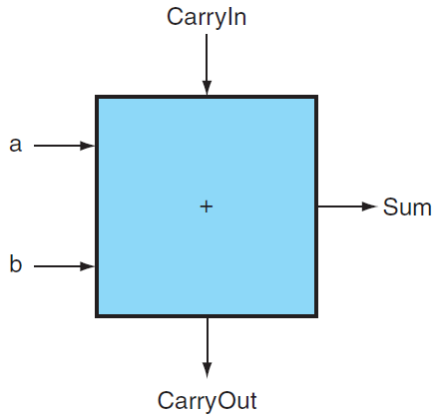
非阻塞赋值示例：

```
always@(posedge clk)
begin
        a <= 1;
        b <= a;
        c  <= b;
end
```

建议使用

A.5 构建基本算数逻辑单元

- 算数逻辑单元：Arithmetic Logical Unit，简称ALU、运算器
  - RV32I的数据位宽为32bit，因此需要构建32bit位宽的ALU
  - 首先构建1bit位宽ALU
- 1位ALU
  - 支持与、或、加法三种逻辑运算



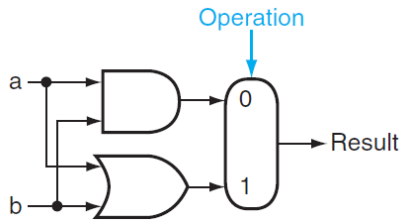
Inputs			Outputs		
a	b	CarryIn	CarryOut	Sum	Comments
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

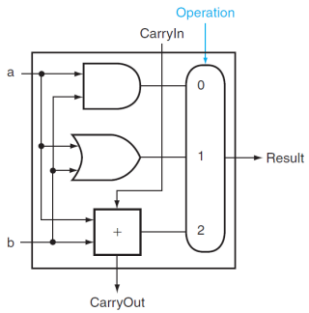
$$\text{CarryOut} = (b \cdot \text{CarryIn}) + (a \cdot \text{CarryIn}) + (a \cdot b)$$

$$\text{Sum} = (a \cdot \overline{b} \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot \overline{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$

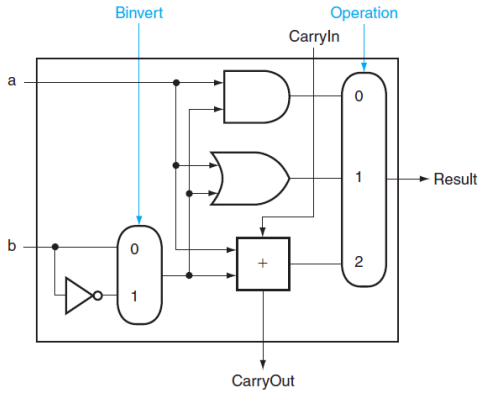
A.5 构建基本算数逻辑单元

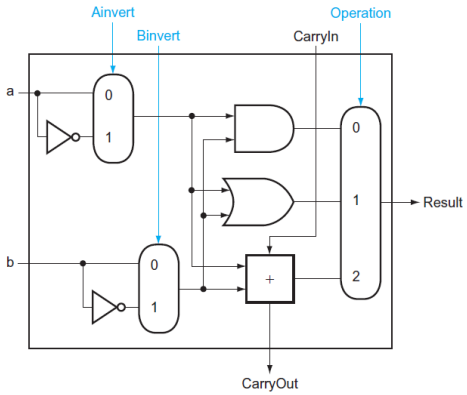
- 1位ALU
  - 支持与、或、加法三种逻辑运算

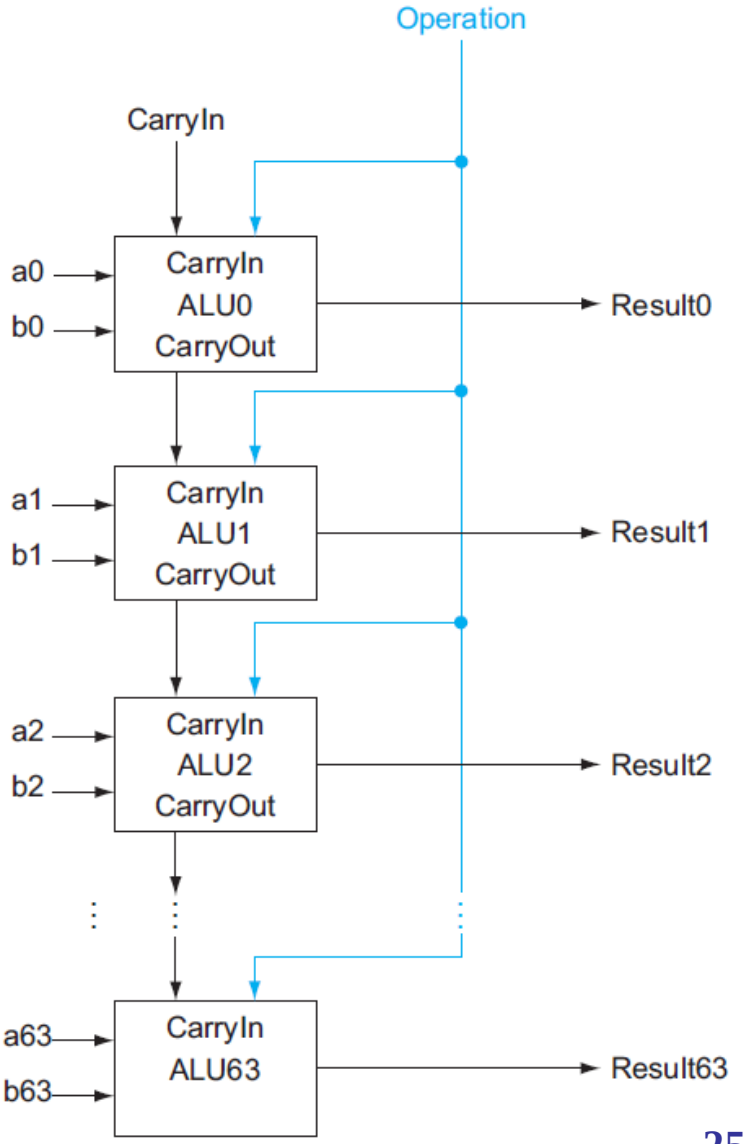




- 64位ALU
  - 将64个1bit ALU依次级联
  - 增加减法操作、或非操作

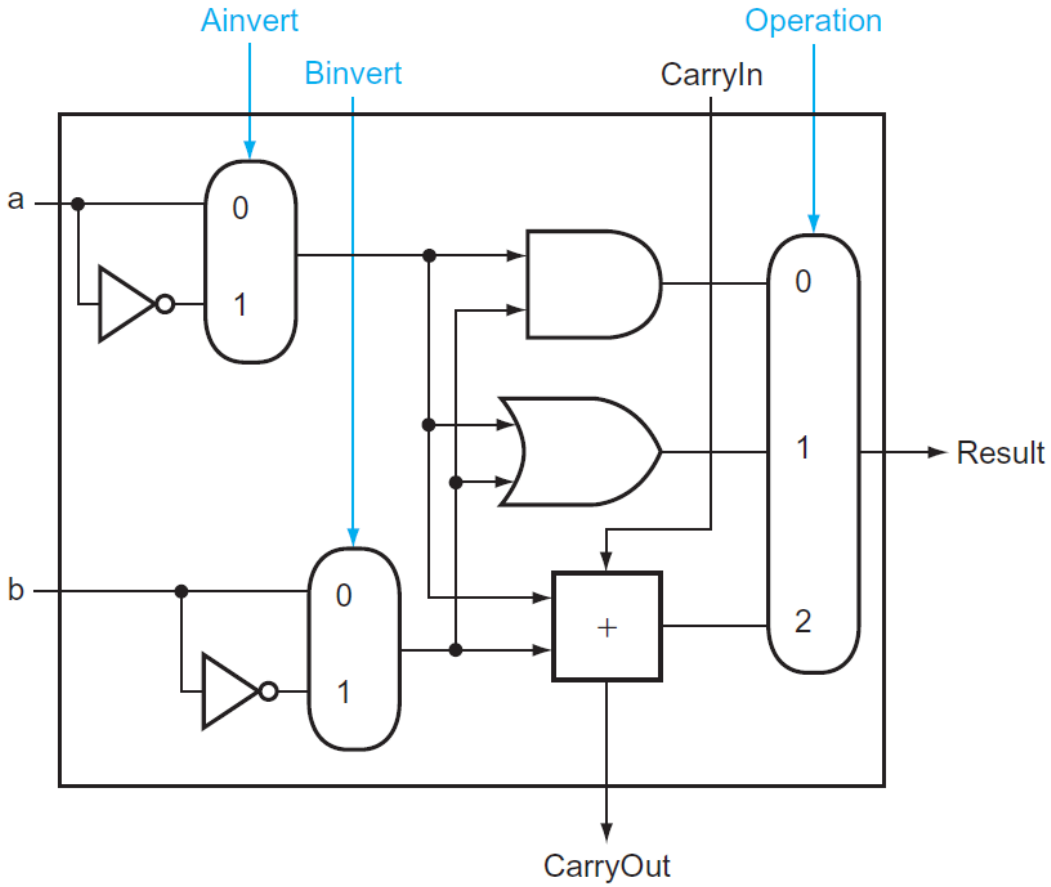






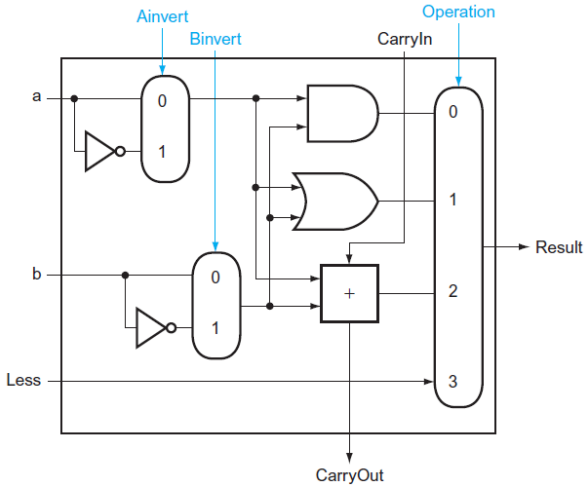
A.5 构建基本算数逻辑单元

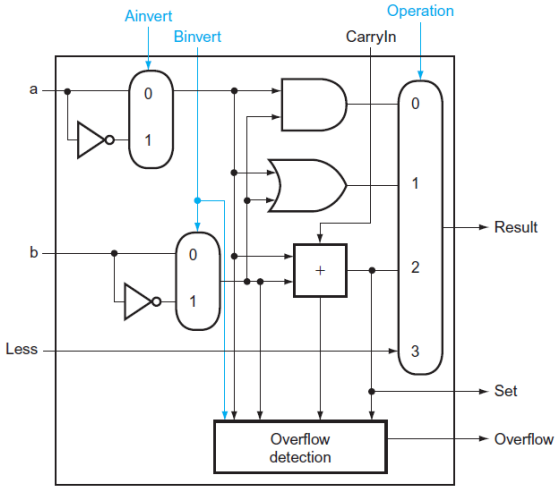
- 请思考：
- 上述64bit ALU已支持加法、减法、与、或、非、或非、与非等操作，还需要支持哪些逻辑操作？

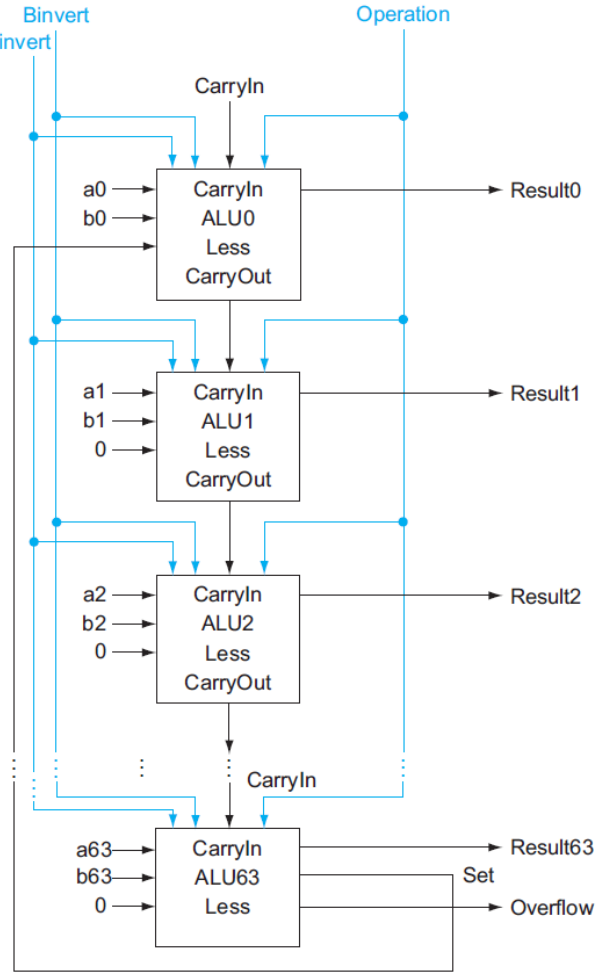


A.5 构建基本算数逻辑单元

- 修改ALU以适应RISC-V
  - 增加小于置位指令（slt，set less than）
  - 如果a<b，则输出1，否则输出0

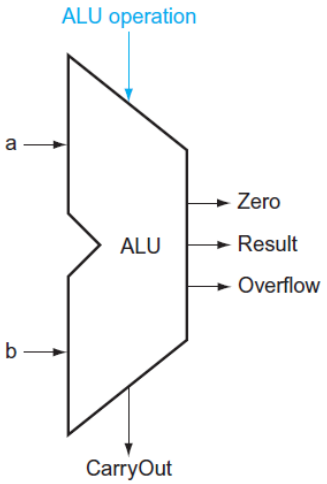




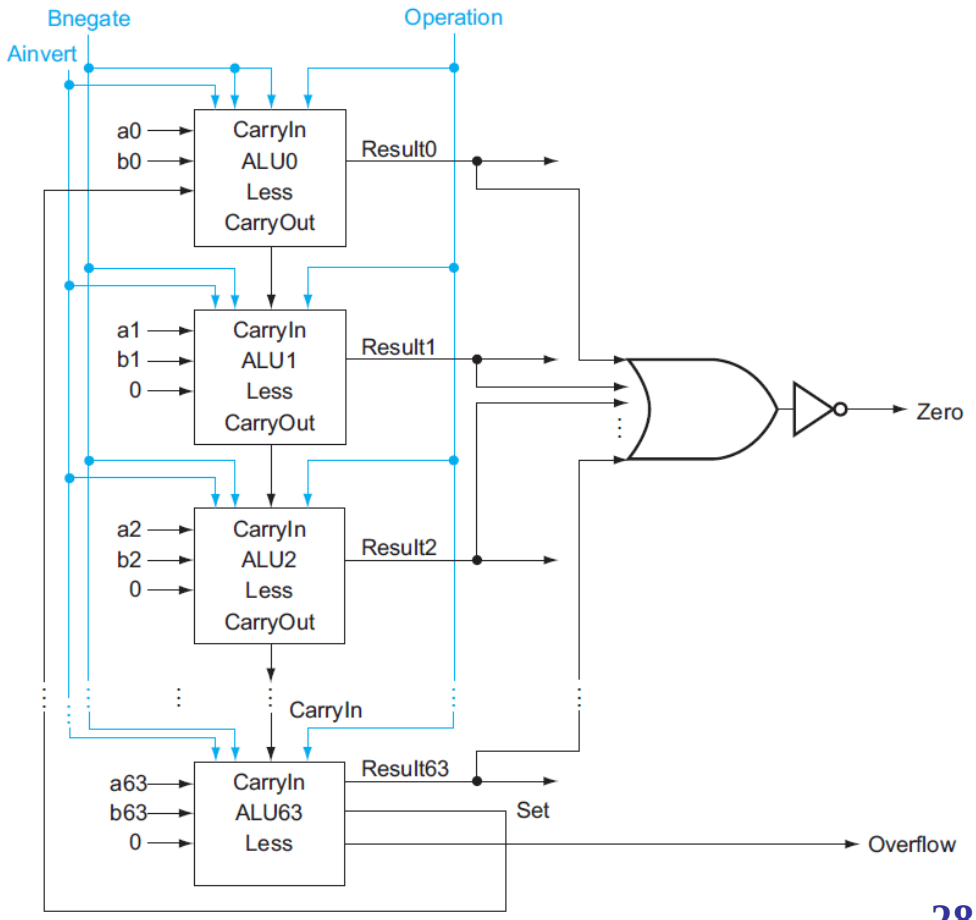


A.5 构建基本算数逻辑单元

- 修改ALU以适应RISC-V
  - 增加相等判断逻辑（beq指令，相等则跳转）



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR



A.5 构建基本算数逻辑单元

- 用Verilog定义RISC-V ALU
  - 纯组合逻辑电路
  - RISCVALU：算数逻辑单元
  - ALUControl：控制信号生成

```
module ALUControl (ALUOp, FuncCode, ALUCtl);
   input [1:0] ALUOp;
   input [5:0] FuncCode;
   output [3:0] reg ALUCtl;
   always case (FuncCode)
   32: ALUOp<=2; // add
   34: ALUOp<=6; // subtract
   36: ALUOP<=0; // and
   37: ALUOp<=1; // or
   39: ALUOp<=12; // nor
   42: ALUOp<=7; // slt
   default: ALUOp<=15; // should not happen
   endcase
endmodule
```

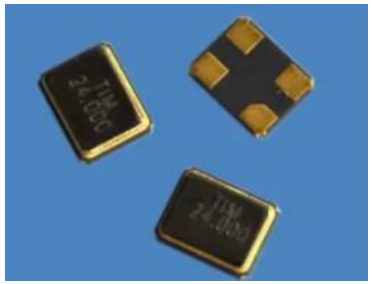
```
module RISCVALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [63:0] A,B;
   output reg [63:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
   always @(ALUctl, A, B) begin //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1 : 0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0;
      endcase
    end
endmodule
```

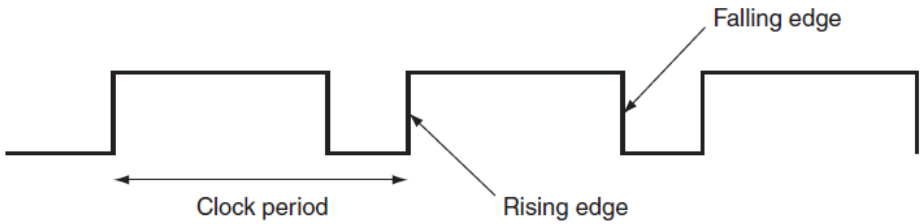
A.6 快速加法：超前进位

- 自学，略

A.7 时钟

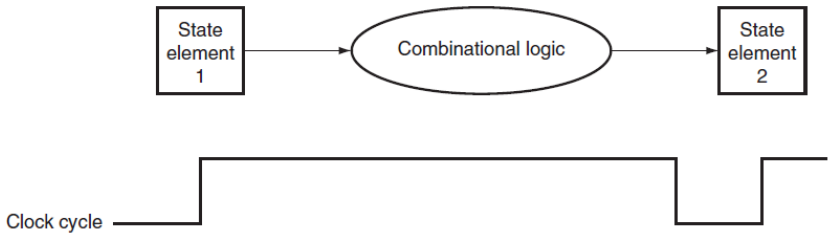
- 定义：是一个具有固定周期的不停翻转的信号
- A clock is simply a free-running signal with a fixed cycle time
- 与时钟相关的参数：周期、频率、占空比、驱动能力、抖动、偏移等
- 一般由专门的器件来生成：有源晶振、无源晶振、锁相环等
- 晶振介绍：
  - https://www.bilibili.com/video/BV1yS4y1k7EE?from=search&seid=10604632539372921070&spm_id_from=333.337.0.0

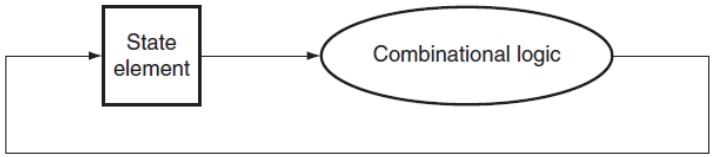




A.7 时钟

- 时序逻辑电路需要通过时钟信号控制何时更新存储元件的状态
  - 电平触发：锁存器
  - 边沿触发：触发器
- 同步系统：在时钟边沿同步更新存储元件状态的电路系统
  - 组合逻辑：（是否是同步系统？）
  - 时序逻辑：锁存器电路（？）、触发器电路（？）
- 边沿触发电路的优点
  - 同步，时序可控
  - 可以实现反馈（有什么用？）

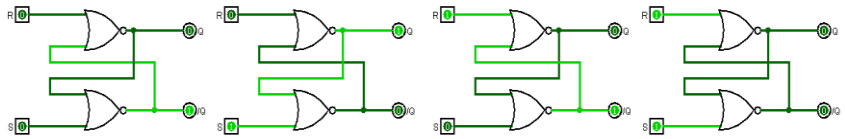




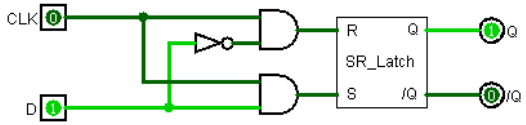
A.8 存储元件：触发器、锁存器和寄存器

- 基本逻辑门 → RS锁存器→ D锁存器→ D触发器→ 寄存器

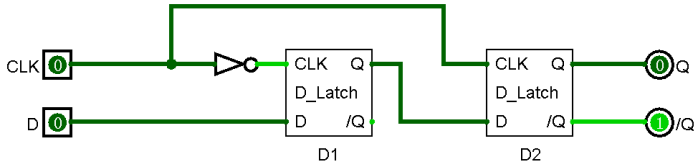
- RS锁存器



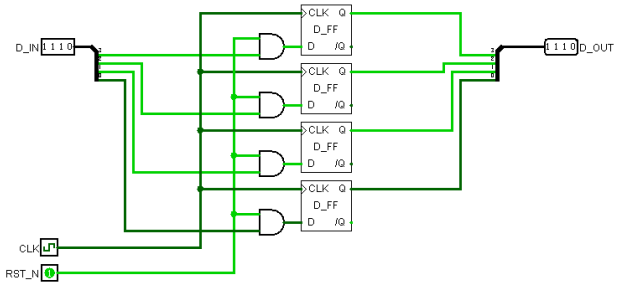
- D锁存器



- D触发器



- 寄存器

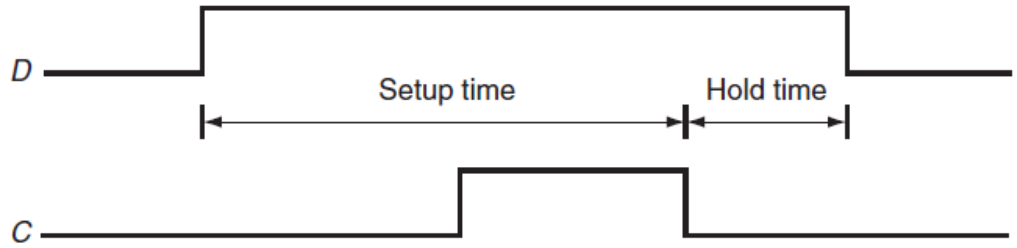


A.8 存储元件：触发器、锁存器和寄存器

- Verilog实现

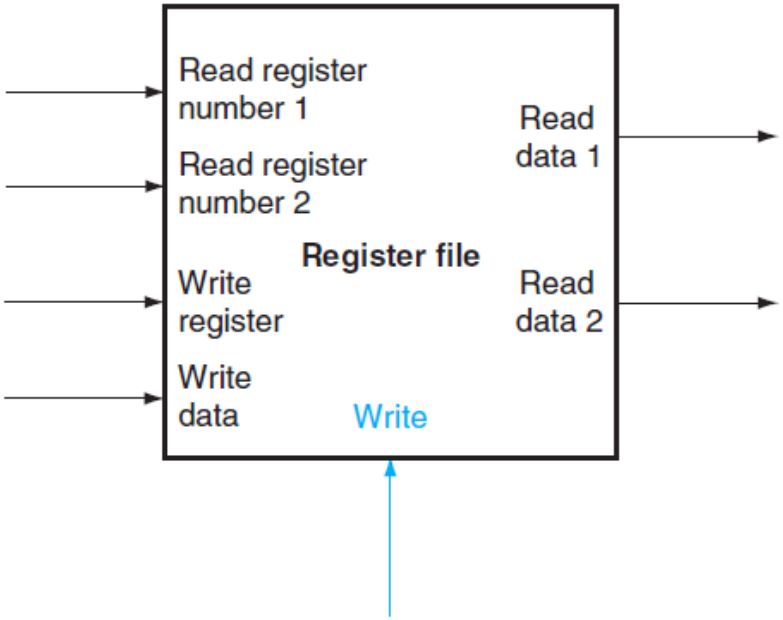
```
module DFF(clock,D,Q,Qbar);
    input clock, D;
    output reg Q;
    output Qbar;
    assign Qbar= ~ Q;
    always @(posedge clock)
       Q=D;
endmodule
```

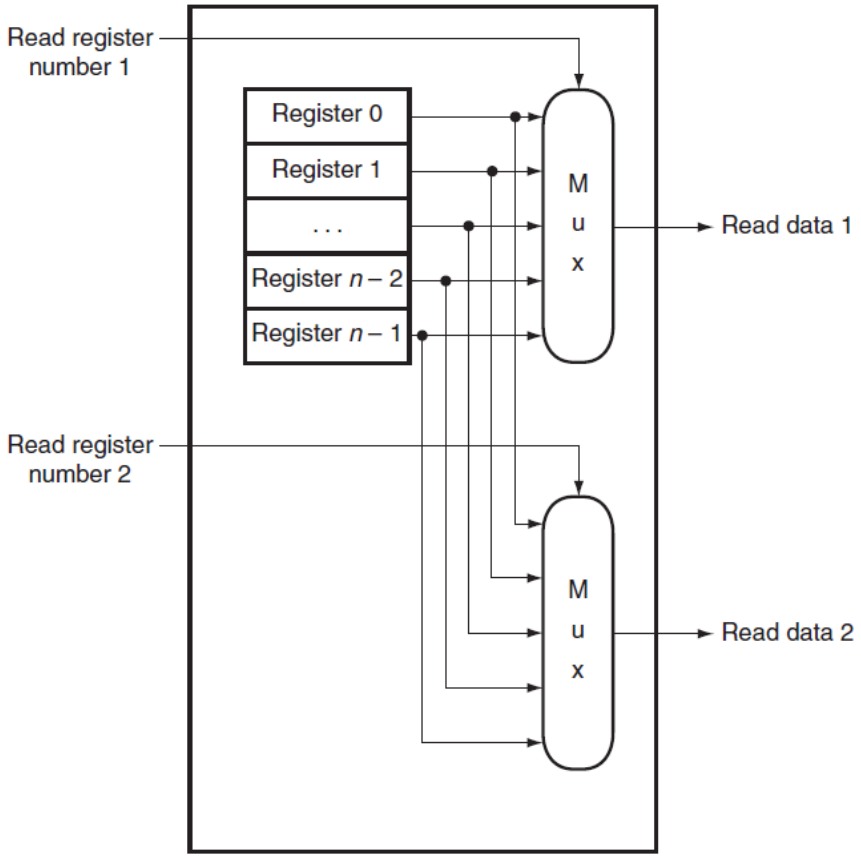
- 建立时间和保持时间



A.8 存储元件：触发器、锁存器和寄存器

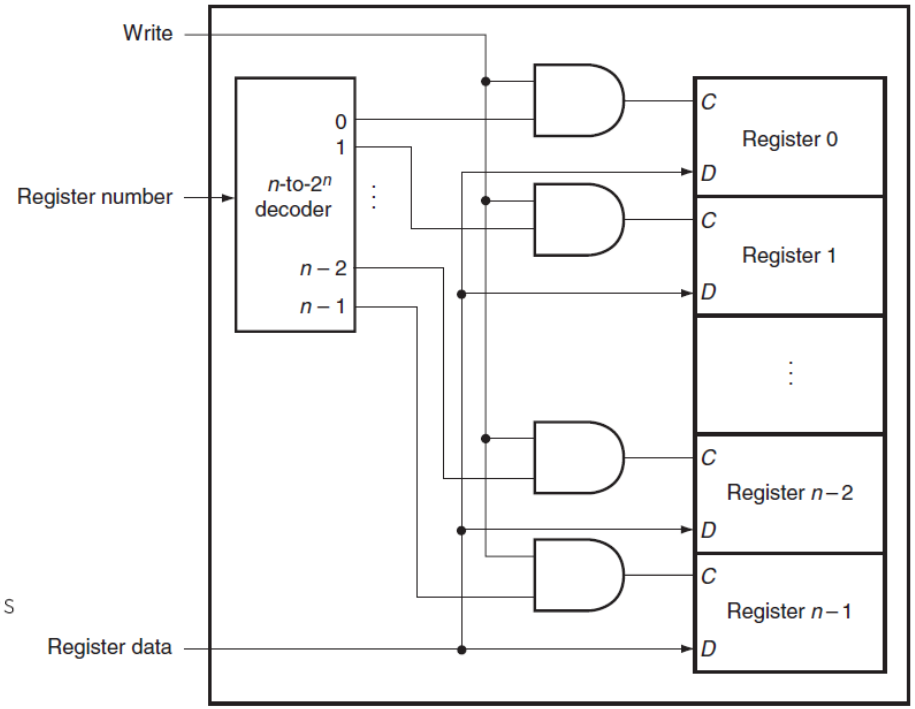
- 寄存器文件（Register File）
  - 在CPU数据通路中至关重要的结构
  - 由一组寄存器组成，可以通过寄存器编号进行读写操作
  - 一般有两组读端口和一组写端口





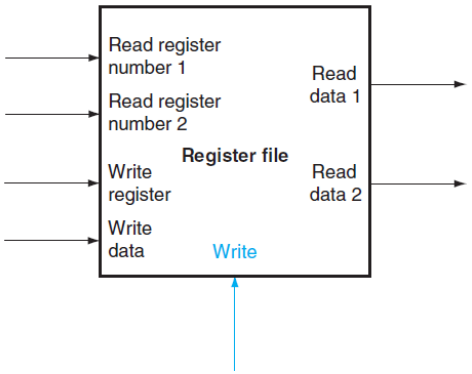
A.8 存储元件：触发器、锁存器和寄存器

- 寄存器文件
  - 写接口结构图



- Verilog实现

```
module registerfile (Read1,Read2,WriteReg,WriteData,RegWrite,
Data1,Data2,clock);

   input [5:0] Read1,Read2,WriteReg; // the register numbers
to read or write
   input [63:0] WriteData; // data to write
   input RegWrite, // the write control
     clock; // the clock to trigger write
   output [63:0] Data1, Data2; // the register values read
   reg [63:0] RF [31:0]; // 32 registers each 32 bits long

   assign Data1 = RF[Read1];
   assign Data2 = RF[Read2];

   always begin
      // write the register with new value if Regwrite is
high
      @(posedge clock) if (RegWrite) RF[WriteReg] <=
WriteData;
   end
endmodule
```



A.8 存储元件：触发器、锁存器和寄存器

Check Yourself

In the Verilog for the register file in Figure A.8.11, the output ports corresponding to the registers being read are assigned using a continuous assignment, but the register being written is assigned in an `always` block. Which of the following is the reason?

a. There is no special reason. It was simply convenient.

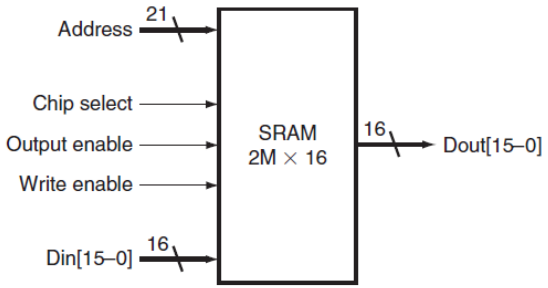
b. Because Data1 and Data2 are output ports and WriteData is an input port.

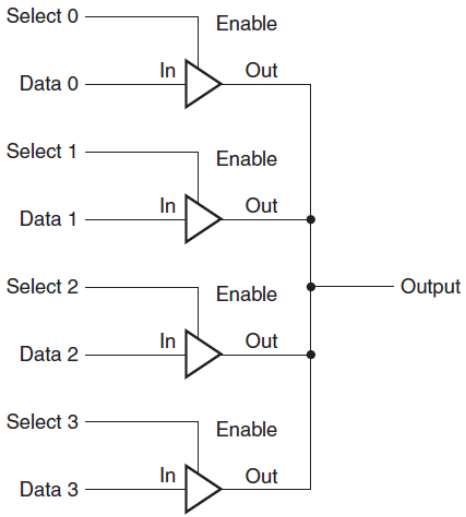
c. Because reading is a combinational event, while writing is a sequential event.

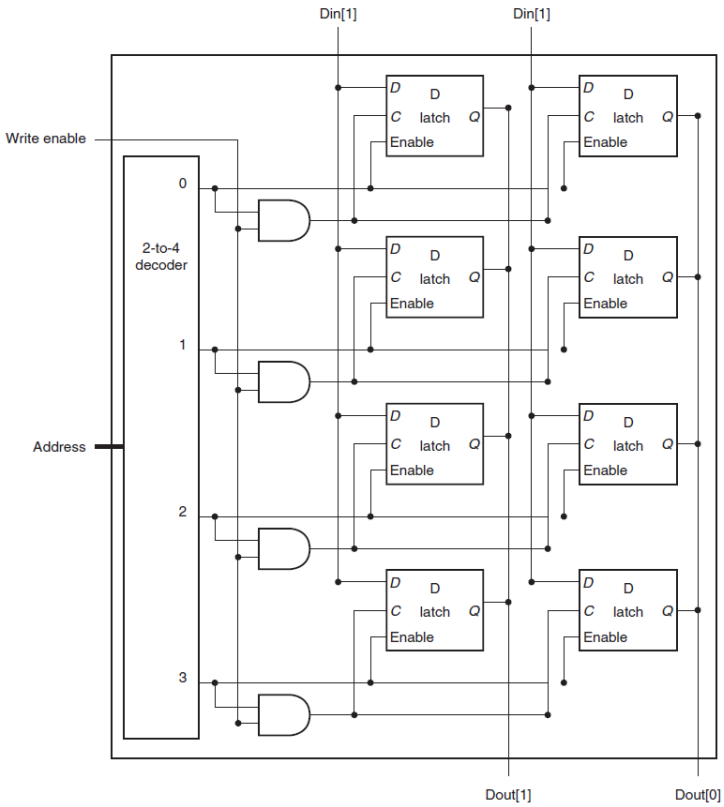
```
module registerfile (Read1,Read2,WriteReg,WriteData,RegWrite,
Data1,Data2,clock);

   input [5:0] Read1,Read2,WriteReg; // the register numbers
to read or write
   input [63:0] WriteData; // data to write
   input RegWrite, // the write control
     clock; // the clock to trigger write
   output [63:0] Data1, Data2; // the register values read
   reg [63:0] RF [31:0]; // 32 registers each 32 bits long

   assign Data1 = RF[Read1];
   assign Data2 = RF[Read2];

   always begin
      // write the register with new value if Regwrite is
high
      @(posedge clock) if (RegWrite) RF[WriteReg] <=
WriteData;
   end
endmodule
```

A.9 存储元件：SRAM和DRAM

- 存储元件
  - 寄存器和寄存器文件的缺点：结构复杂、成本高、集成度低
  - 大容量数据存储需要使用SRAM（静态）、DRAM（动态）
- SRAM：static random access memories
  - 巨型多路选择器不具备可行性
  - SRAM使用共享输出线的方式实现

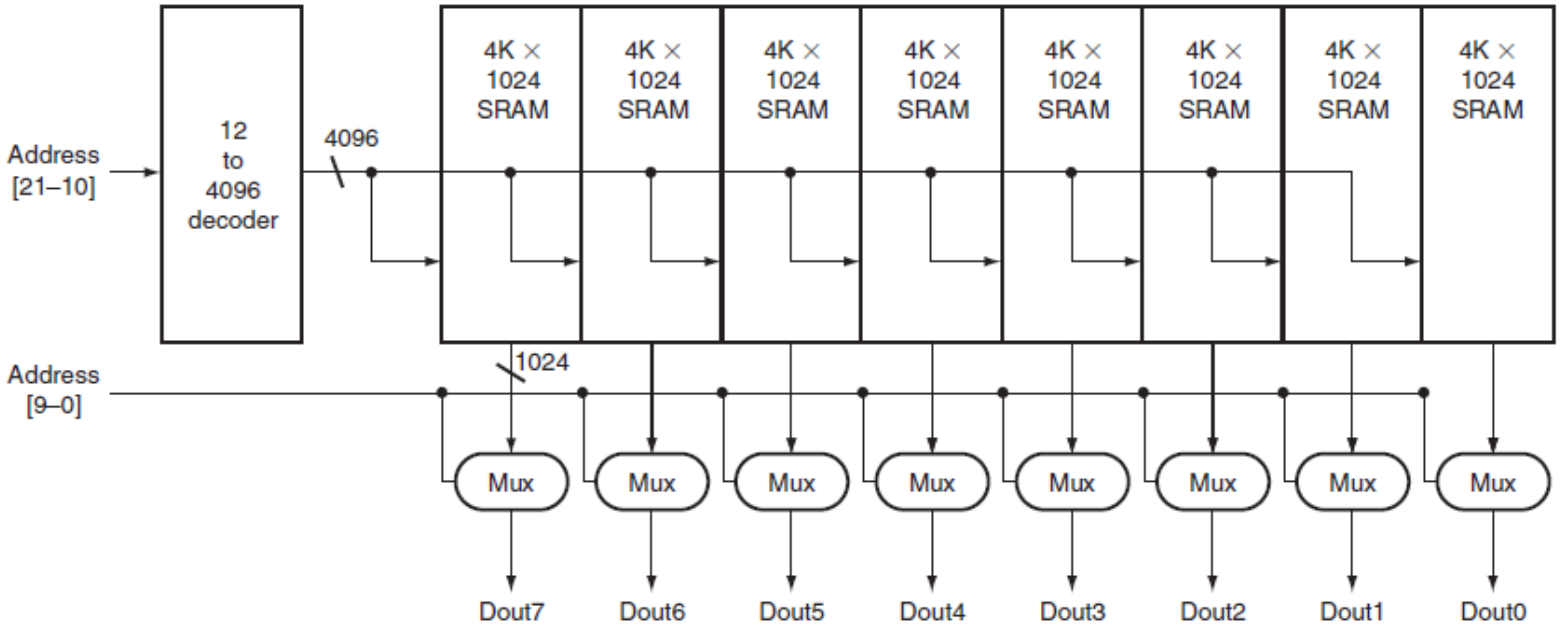






A.9 存储元件：SRAM和DRAM

- SRAM：static random access memories
  - 前面例子的设计中消除了巨型选择器的需求
  - 仍然需要大型译码器和大量字线
  - 改进：使用二级译码装置

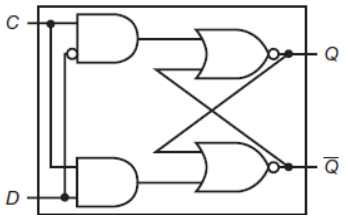


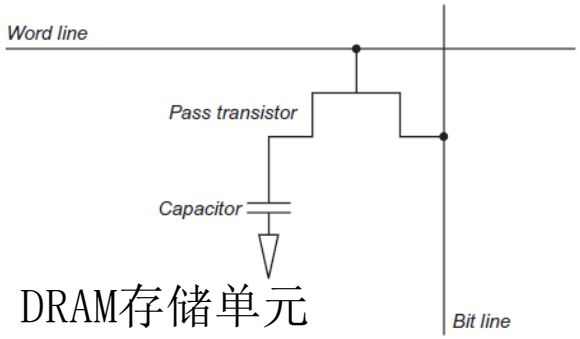
  - 同步SRAM：SSRAM，簇发传输，由时钟信号控制

A.9 存储元件：SRAM和DRAM

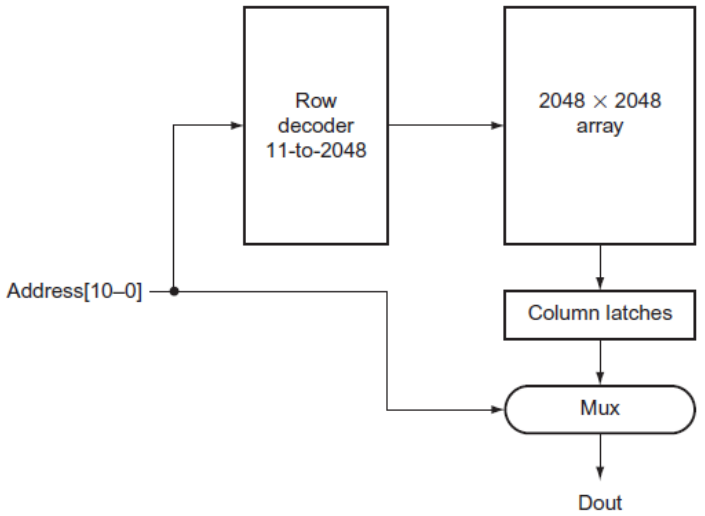
- DRAM：Dynamic Random Access Memory
  - 中文全程：动态随机访问存储器
  - SRAM数据保存：双稳态电路（4~6个晶体管/bit）
  - DRAM数据保存：电容(1个晶体管/bit)
  - 优点：结构简单、成本更低，集成度更高
  - 缺点：电容漏电，需要定时刷新（动态）
- SDRAM
  - 同步动态随机访问存储器
  - Synchronous DRAM
- DDR SDRAM
  - 双倍数据速率SDRAM
  - Double Data Rate SDRAM

SRAM存储单元





DRAM存储单元



A.9 存储元件：SRAM和DRAM

- 最新主流DDR SDRAM相关参数
  - 外形规格：双列直插式存储模块 Dual-Inline-Memory-Modules，简称 DIMM
  - 频率：时钟1600MHz，数据3200MHz
  - 容量：128G
  - 价格：约30元/GB（消费级）

现代海力士（SK hynix） DDR4 ECC RDIMM REG 工作站 服务器内存条 LMKJ 128G DDR4 3200 REG 服务器内存

海力士原厂DDR4-RDIMM-REG服务器内存条，不支持笔记本台式机！可兼容（联想-IBM-戴尔-惠普-浪潮-华为-思科-华3等服务器）

京东价 ¥ 7999.00 降价通知

累计评价 12

优惠券 满100减5

京东超市 金士顿（Kingston）128GB USB3.2 Gen 1 U盘 DTX 时尚设计 轻巧便携

【金士顿装机盛典】晒单赢50元E卡，会员专享百元券包，更有Kingston.Fury定制滑板等你来赢！请戳~查看>

甜蜜礼 心意之选，爱耀出色

京东价 ¥ 74.90 降价通知

¥ 69.90 粉丝价 关注店铺，即享粉丝价

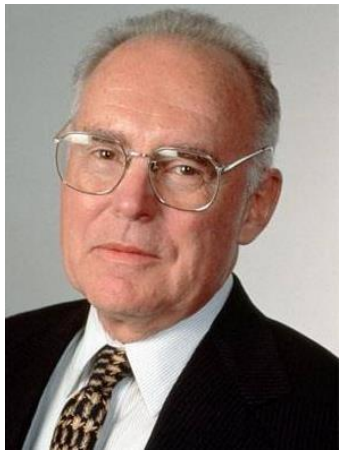
A.9 存储元件：SRAM和DRAM

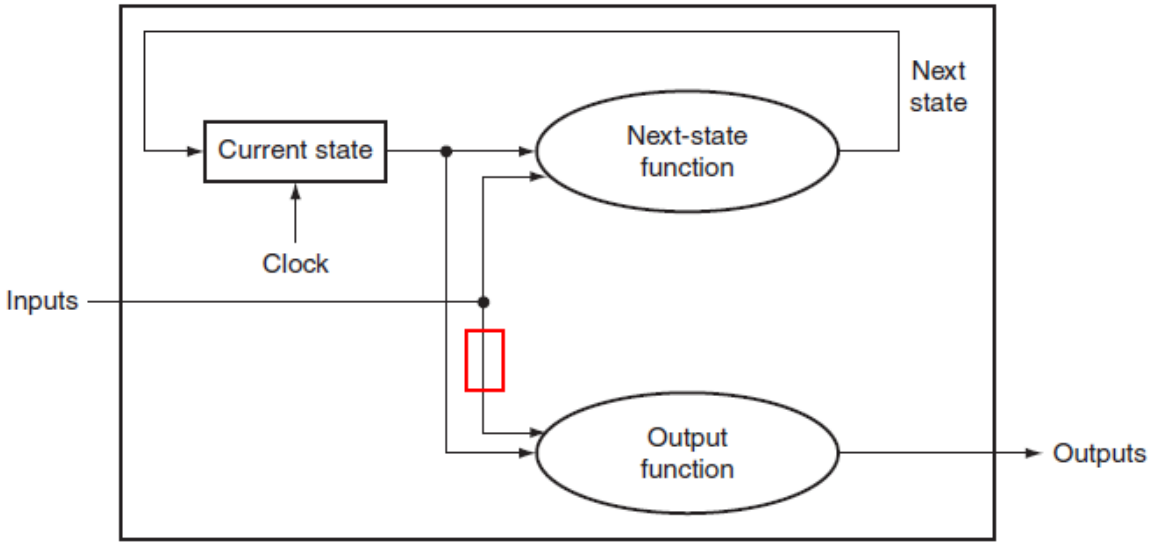
- 错误修正
  - 大容量存储器中存在数据损坏可能性
  - 使用校验码来进行检测或修正
  - 奇偶校验（奇校验、偶校验）：简单，只能检测奇数位的错误
  - ECC：error correction codes，又称汉明码/海明码/Hamming code
    - 根据校验位的数值，可以判定是否出错，以及出错位

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverage	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

A.10 有限状态机

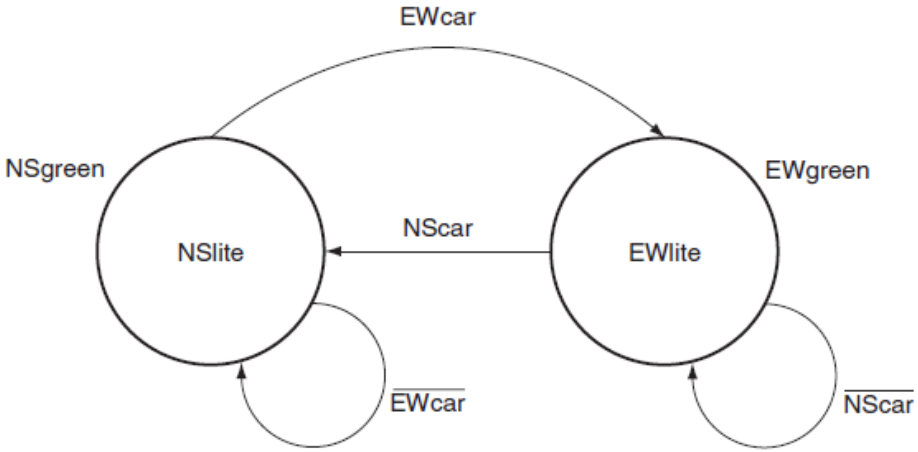
- 有限状态机：FSM，**F**inite-**S**tate **M**achines
- 一种时序逻辑函数，包括一组输入、输出、将当前状态和输出映射到新状态的下一状态函数，以及将当前状态和输入映射到一组有效输出的输出函数
  - 摩尔型：输出依赖于当前状态，时序更好
  - 米莉型：输出依赖于当前状态和当前输入，状态更少
- 是实现时序逻辑控制的重要手段





A.10 有限状态机

- 例：交通灯
  - 输入信号：
    - EWcar：东西方向有车
    - NScar：南北方向有车
  - 输出信号
    - EWlite：东西方向灯，1为绿灯，0为红灯
    - NSlite：南北方向灯，1为绿灯，0为红灯
  - 两个状态：
    - EWgreen：东西方向通行
    - NSgreen：南北方向通行
  - 要求：
    - 每30秒更新一次状态：跳转或保持
  - 建议：FSM采用三段式写法
    - 参考VerilogOJ平台第56题




```
module TrafficLite (EWCar,NSCar,EWLite,NSLite,clock);

   input EWCar, NSCar,clock;
output EWLite,NSLite;

reg state;

initial state=0;  //set initial state

//following two assignments set the output, which is based
only on the state variable
assign NSLite = ~ state; //NSLite on if state = 0;
assign EWLite = state; //EWLite on if state = 1

always @(posedge clock) // all state updates on a positive
clock edge
   case (state)
      0: state = EWCar; //change state only if EWCar

      1: state = ~ NSCar; // change state only if NSCar

   endcase
endmodule
```

A.11 定时方法

- 略

A.12 现场可编程设备

- 略

A.13 本章小结

- 略