

- 数字系统的**RTL**设计
- 学时分配：**10**

# 目的和目标

- 初步了解复杂数字系统的“数据通路+控制逻辑”的设计思想；
- 初步掌握数据流模型的建立和描述；
- 初步了解数字系统模块划分和时序设计的方法。
- 掌握简单CPU(Mu0, 8051兼容CPU)的设计方法；
- 了解IT产业链中核心芯片(CPU等)的高性能设计所面临的机遇和挑战。

# 说明

- 复杂数字系统的设计应该先经过系统级验证，然后才转换成数据流模型。
- 系统级建模和验证的方法很多，也没有统一的模型。公认最有潜力的描述语言是**SystemC**。
- 限于课时，本部分课件省去系统级的建模和验证，只讲述数据流模型的设计和描述。

# 内容安排

- 硬件的**RTL**模型的特点。
- 数据通路与控制逻辑。
- 一个简单的数据处理模块。
- 曼彻斯特处理器**MU0**。
- 流水线与超标量**CPU**设计简介。

# 复杂数字系统的RTL设计思想

- 数字系统（RTL级）设计

— ● Data Path

● 运算单元(组合电路)

● 存储单元(寄存器)

— ● Control Logic

● 有限状态机(组合部分和时序部分)

● 微码ROM描述



# 硬件的RTL模型的特点

1. **RTL**模型中的信号代表了硬件中数据的实际移动方向以及电路的互连关系；
2. **RTL**模型中的语句与实际寄存器的结构模型之间存在直接的映射关系；
3. **RTL**模型指定了寄存器级的电路元件之间的连接关系，从而隐藏了电路结构；
4. **RTL**模型指定了存储单元的复用结构及总线；
5. **RTL**模型中明确指定了各个寄存器的驱动时钟；
6. **RTL**模型中通常不采用抽象的数据类型

# 前期知识准备

- 存储器: **RAM**, **ROM**

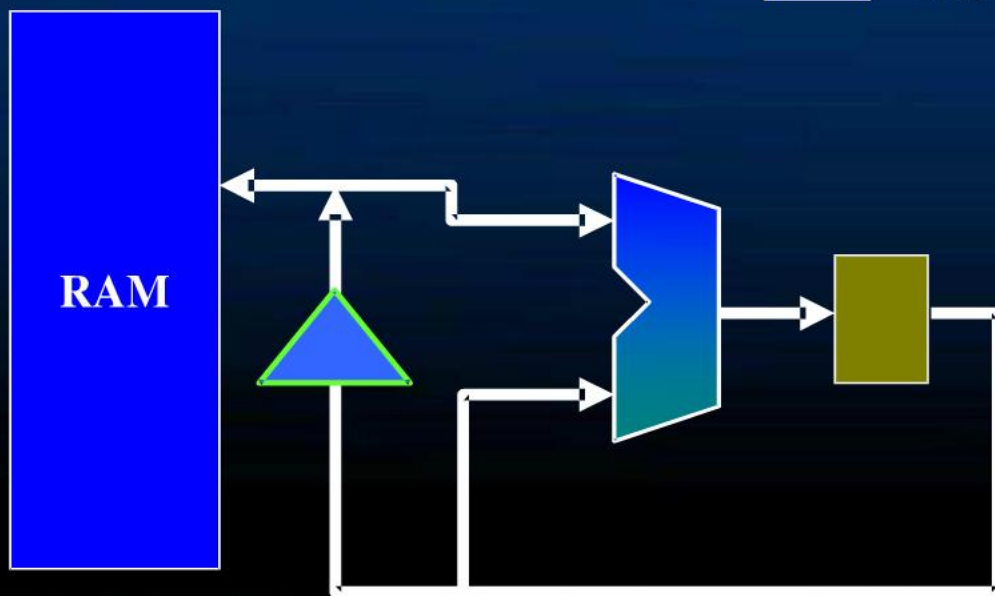
# 设计实例





# 数据通路

时序电路  
组合电路



Addr

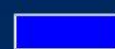


RAM

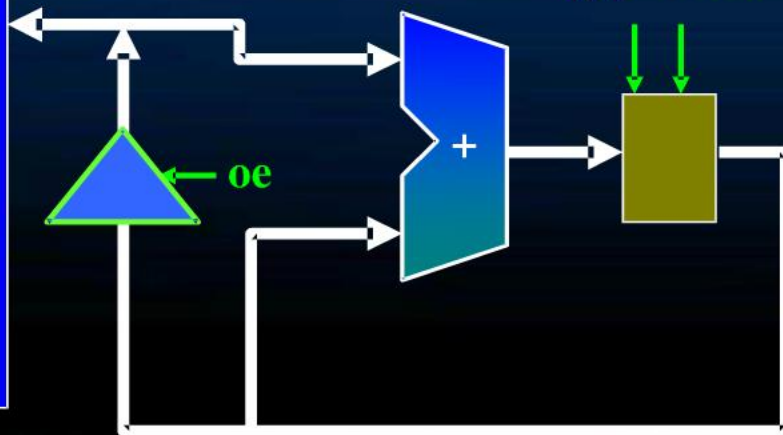
RnW MRq



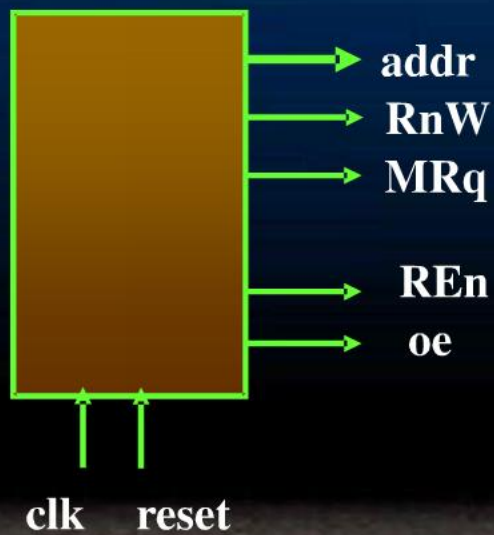
时序电路



组合电路



# 控制逻辑



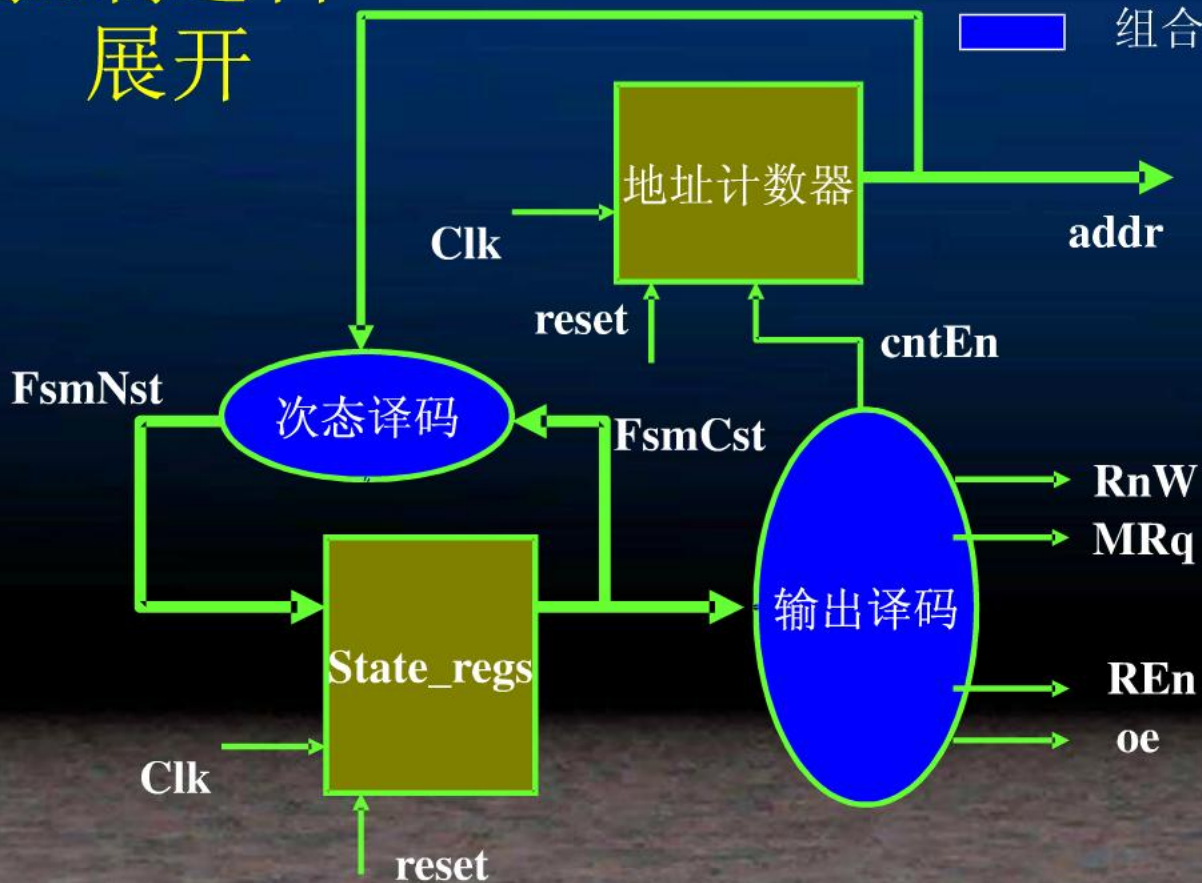
# 时序设计

一定注意，一般  
时序设计和状态  
图设计应该在代  
码设计之前



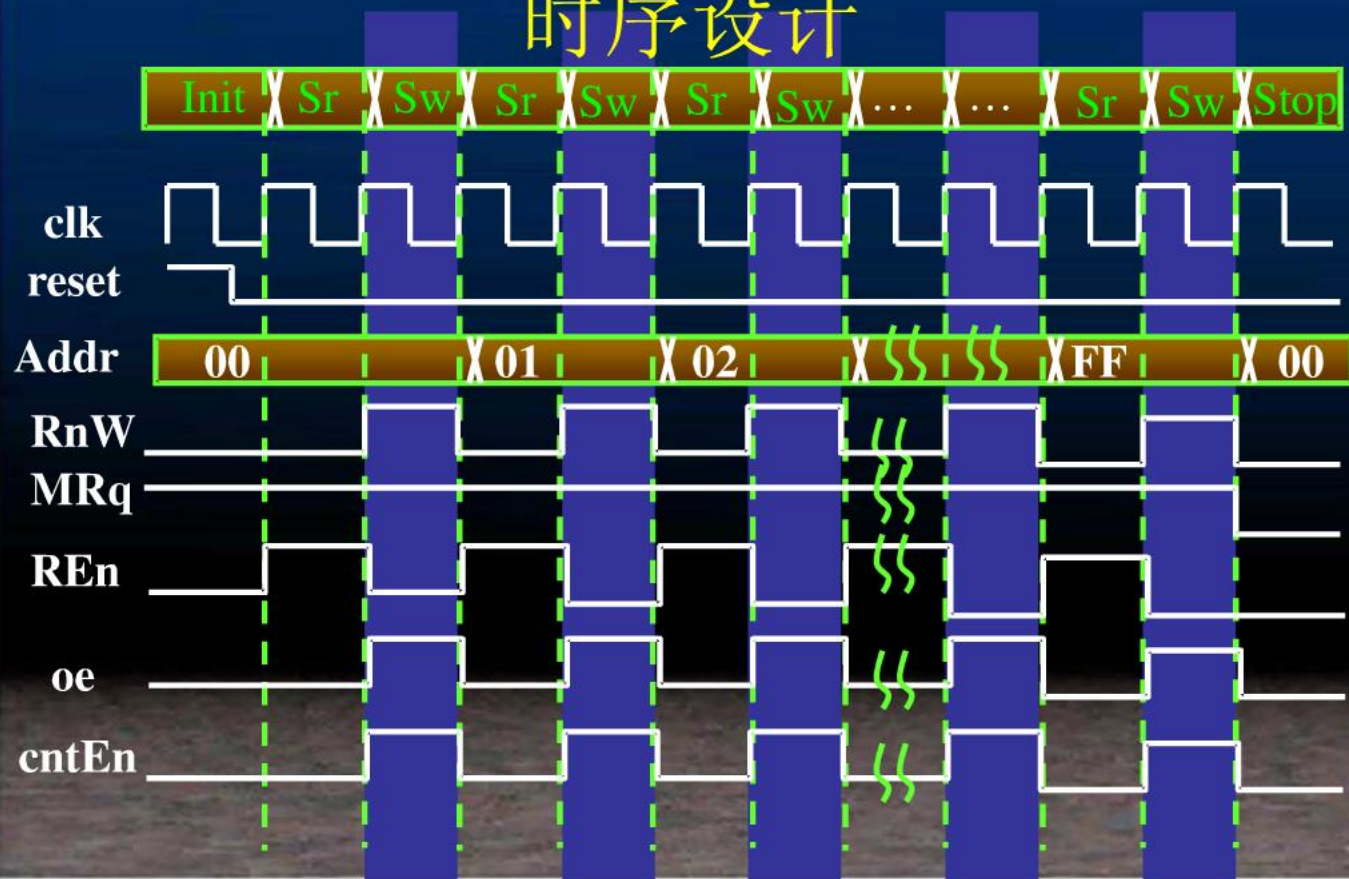
# 控制逻辑展开

时序电路  
组合电路

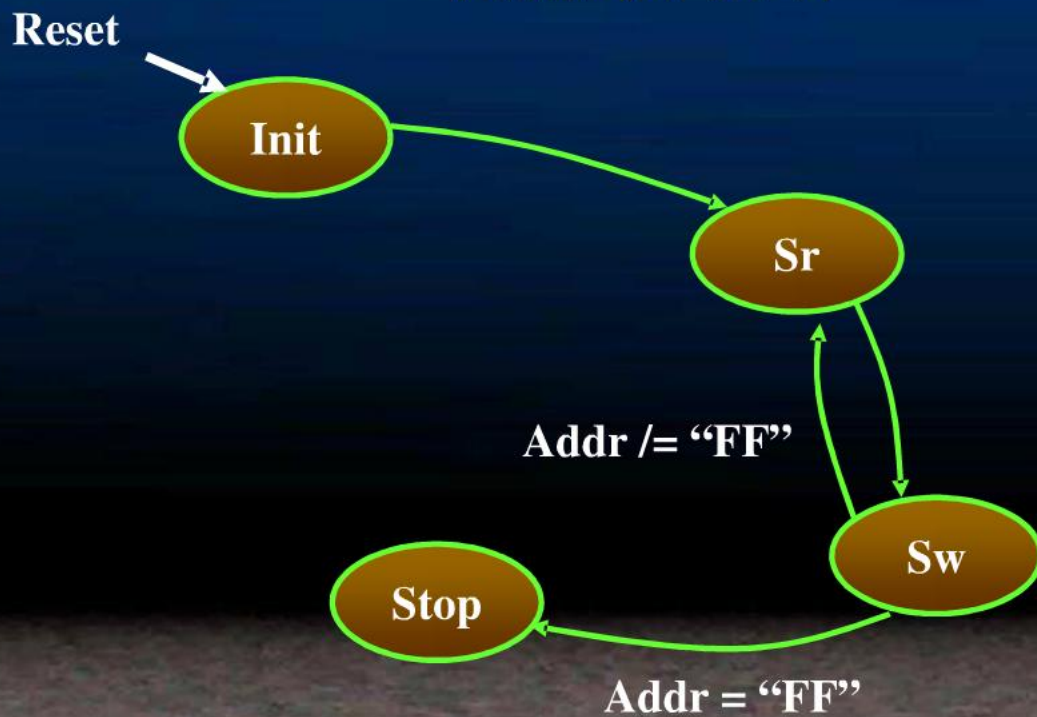




# 时序设计



# 状态机设计



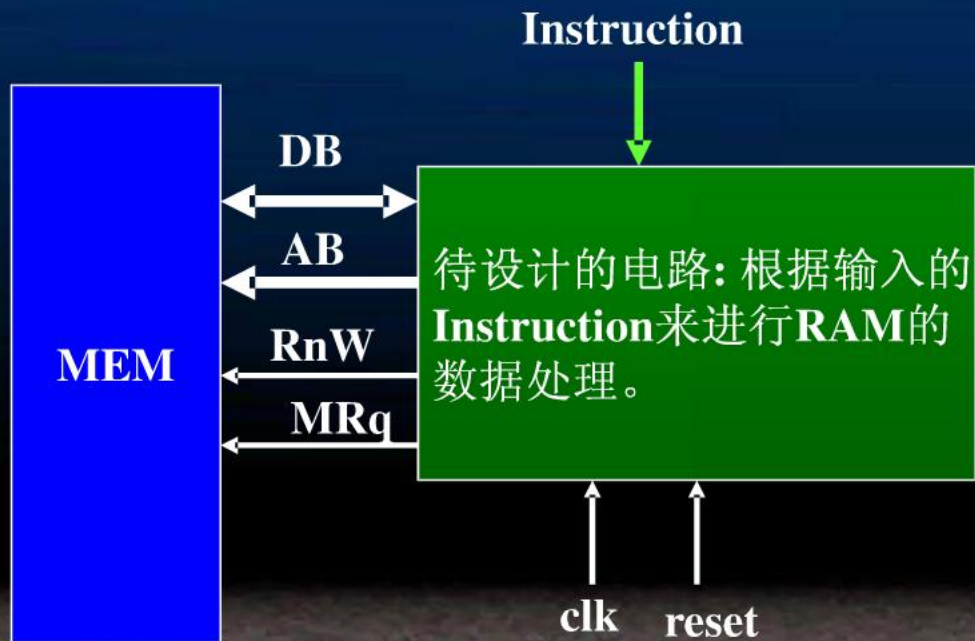
## 状态转换表

Cst	Addr	Nst	RnW	MRq	REn	Oe	CntEn
Init	x	Sr	0	1	0	0	0
Sr	x	Sw	0	1	1	0	1
Sw	FF	Stop	1	1	0	1	1
Sw	其他	Sr	1	1	0	1	1
Stop	x	Stop	0	0	0	0	0

# 代码描述

- 略。
- 总结：一般的数字系统设计中，
  - 第一步，系统需求分析；
  - 第二步，系统级建模及验证；
  - 第三步，模块接口及时序设计
    - 数据通路
    - 控制逻辑
  - 第四步，代码编写；
  - 之后，进行正常的设计迭代。

# 需求修改...





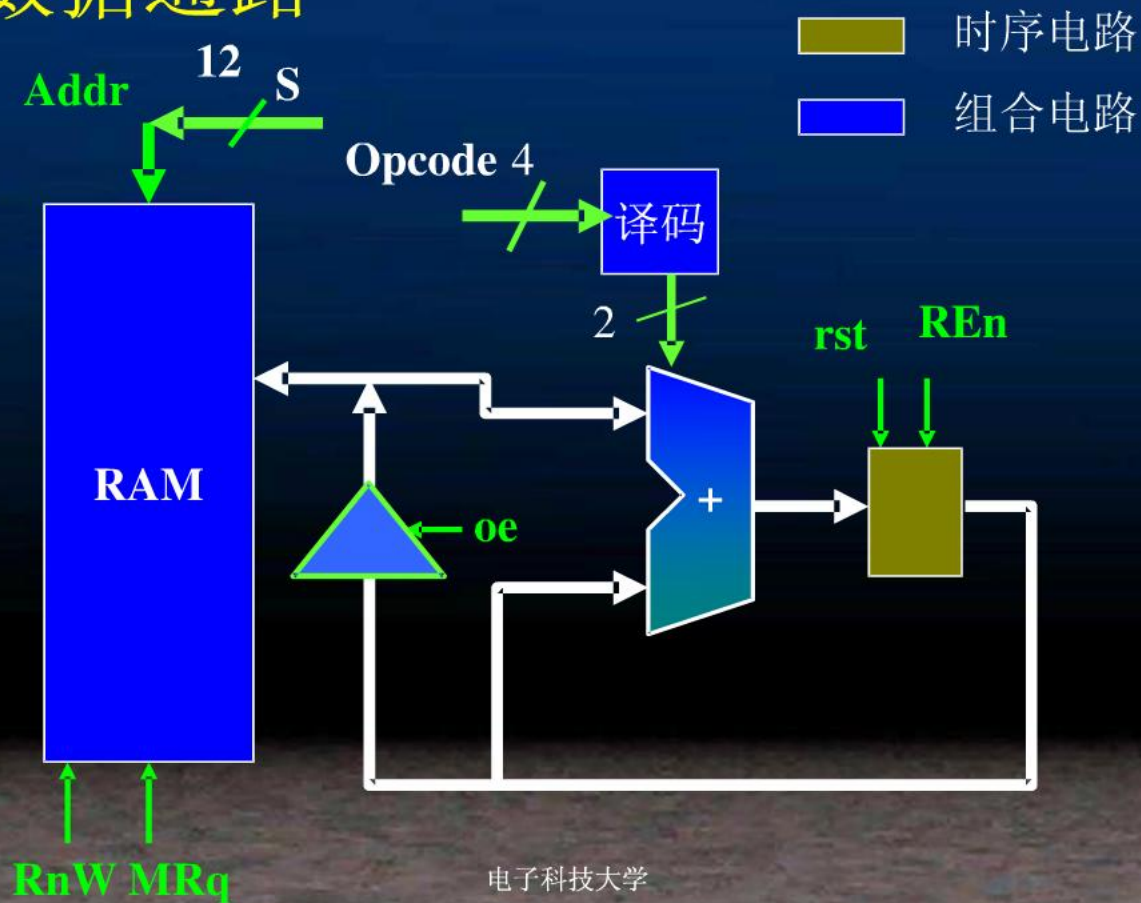
# Instruction定义

15          12   11                                  0



Opcode	S	
0000	...	$ACC := mem_{16}[S]$
0001	...	$mem_{16}[S] := ACC$
0010	...	$ACC := ACC + mem_{16}[S]$
0011	...	$ACC := ACC -f mem_{16}[S]$
0111	...	stop

# 数据通路



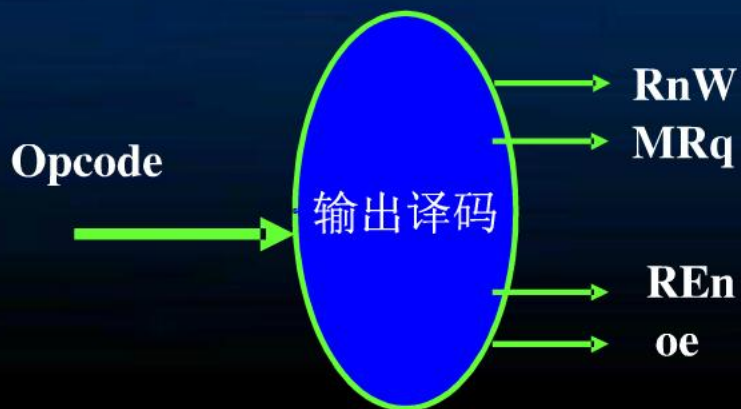
# 时序设计



注意：这里假设某些外部电路保证  
**Opcode**在  
**Reset**  
时为**0110**，  
其信号变化被**clk**同步。

# 控制 逻辑

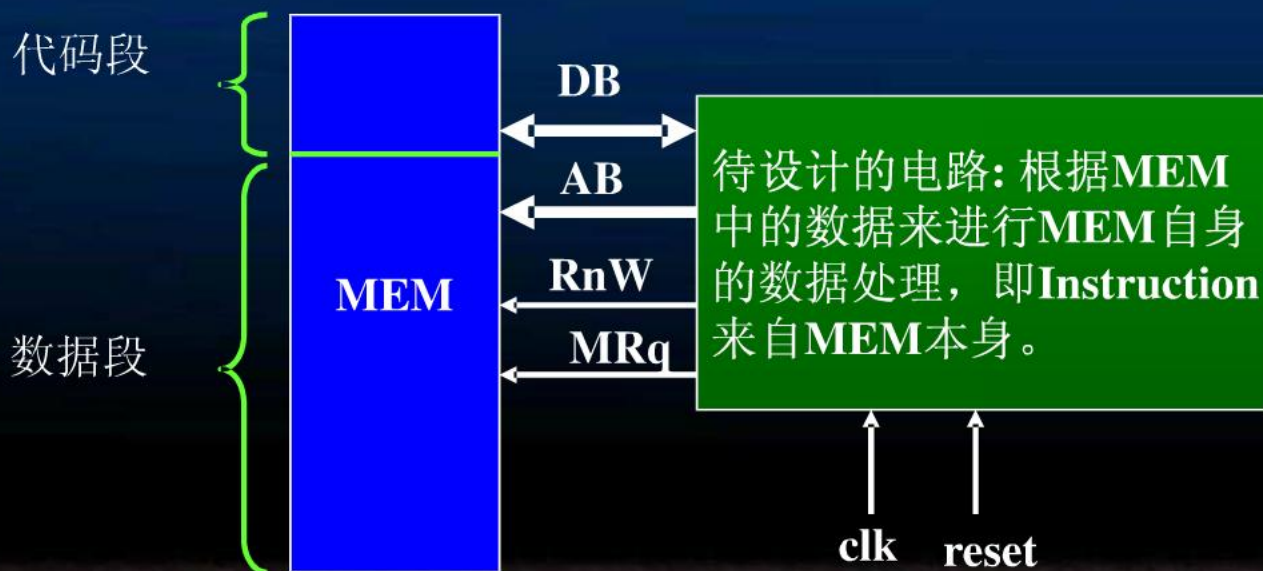
■ 时序电路  
■ 组合电路



**reset**

电子科技大学

## 需求升级...





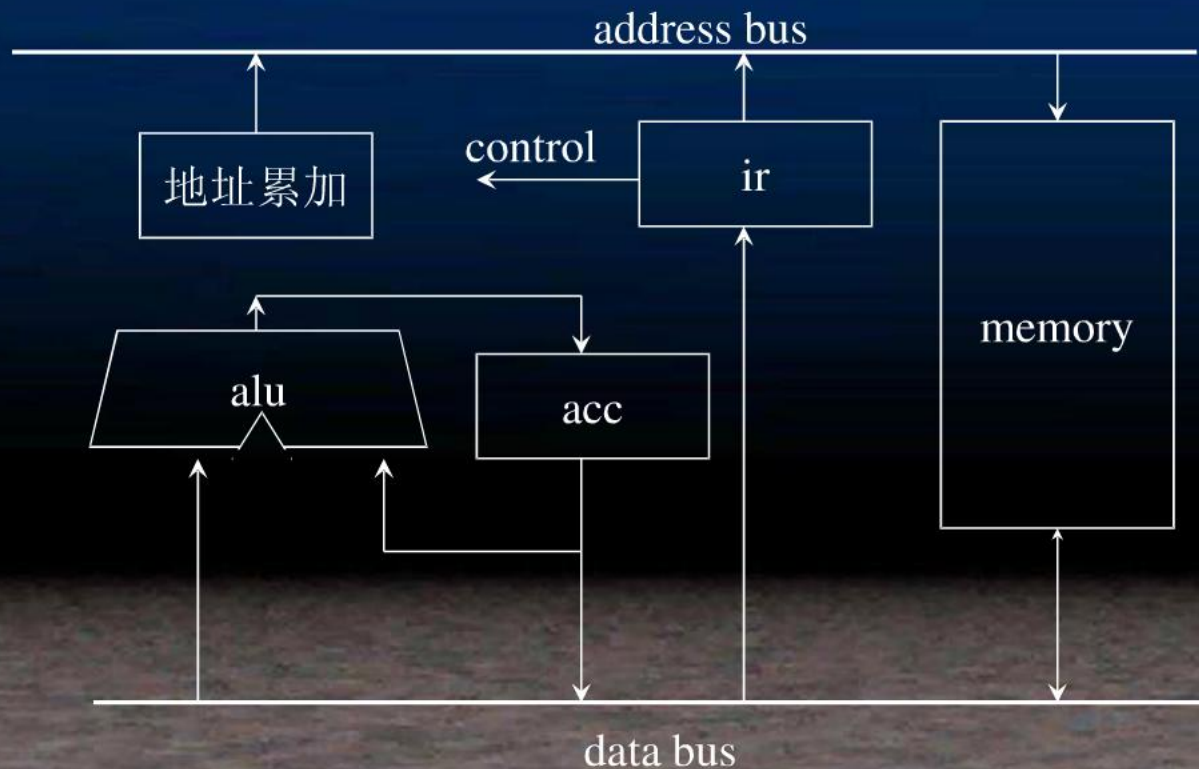
# 数据通路修改分析

- 存储器的访问分为两个阶段：
  - 1.**Instruction**的读取----取指。
  - 2.数据的读取----执行。
- 分析地址和数据总线的占用：
  - 1.取指阶段：**Instruction**的地址+**Instruction**。
  - 2.执行阶段：操作数的地址+操作数。
- 需要解决的问题：
  - 指令和数据的冲突避免；
  - 地址产生机制。

# 问题的解决

- 指令和数据的冲突避免：
  - 缓存指令：引入**IR**寄存器；
- 地址的产生：
  - 指令地址：每执行一次指令就加**1**；
  - 操作数地址：由**IR**寄存器的**S**域产生。

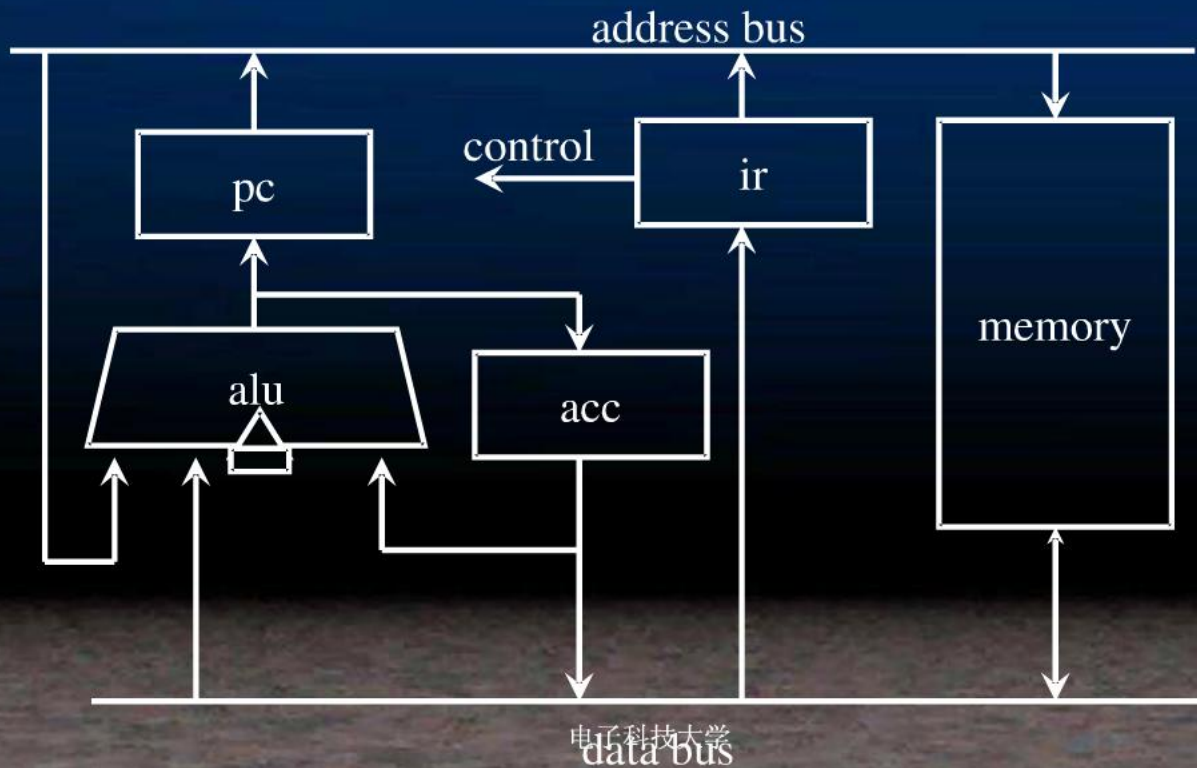
# 数据通路(省略总线复用)



## 进一步硬件优化

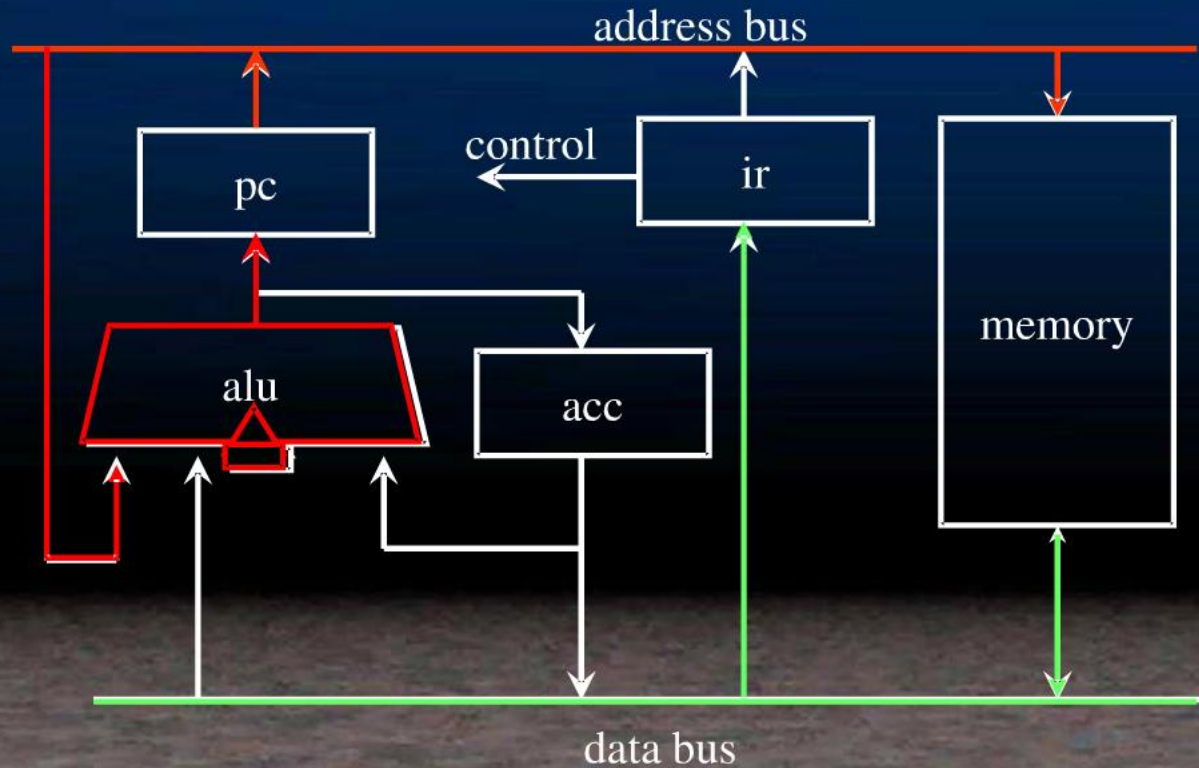
- 发现ALU在取指阶段空闲，执行阶段工作，则可以考虑让ALU在取指阶段对指令地址进行累加，从而使得硬件更加充分利用。
- 最终的指令周期设计：
  - 取值阶段：取指令，同时指令地址+1。
  - 执行阶段：取操作数，运算。

# 最终数据通路(简化)

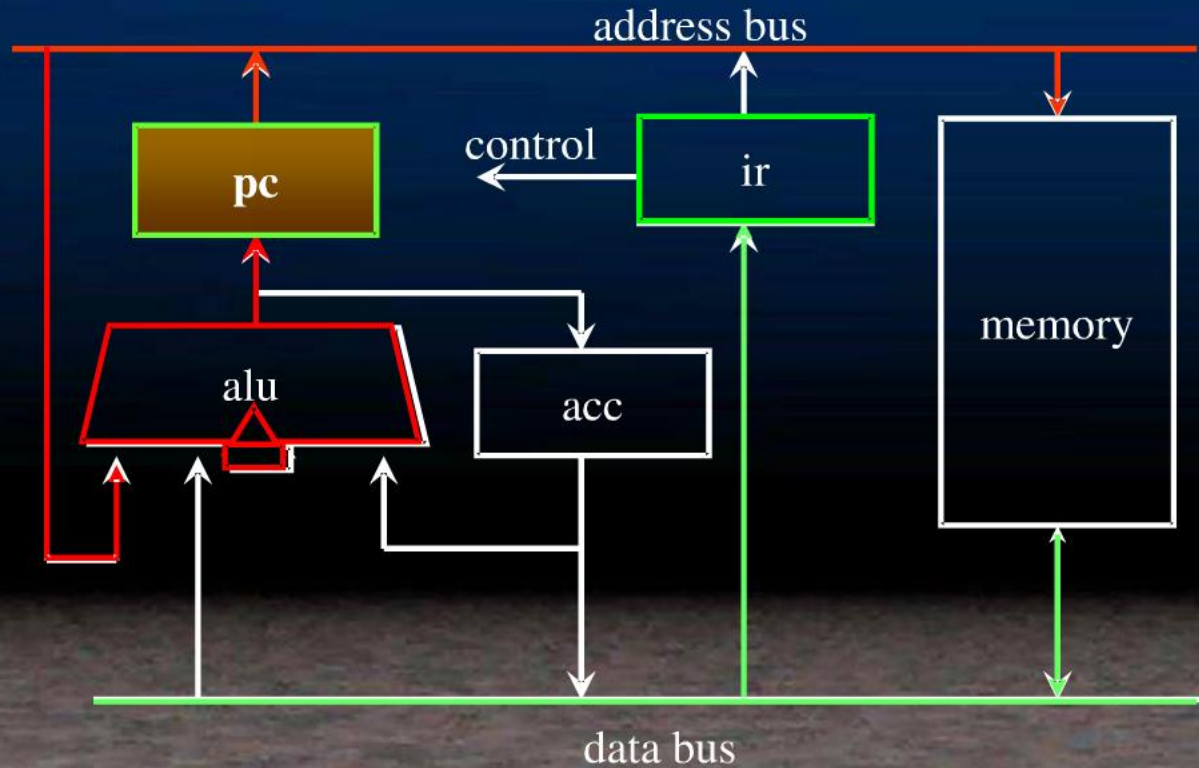




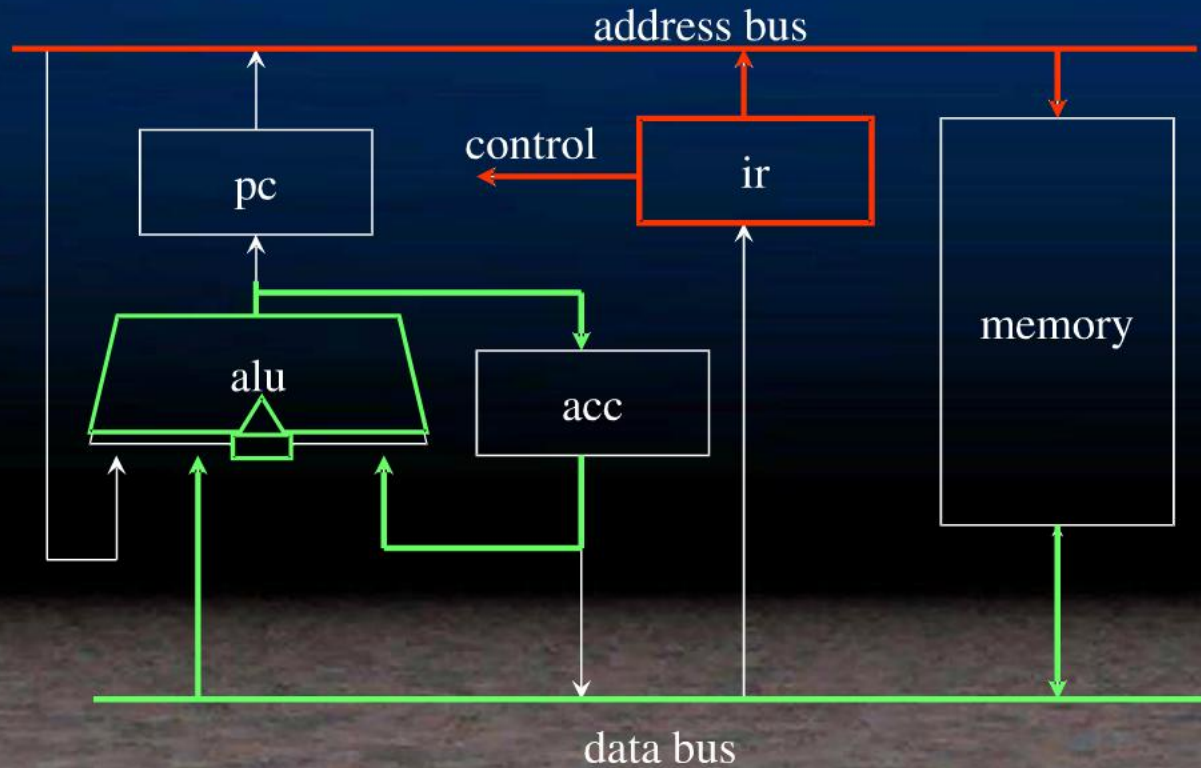
# Cycle1: 取指 & PC++



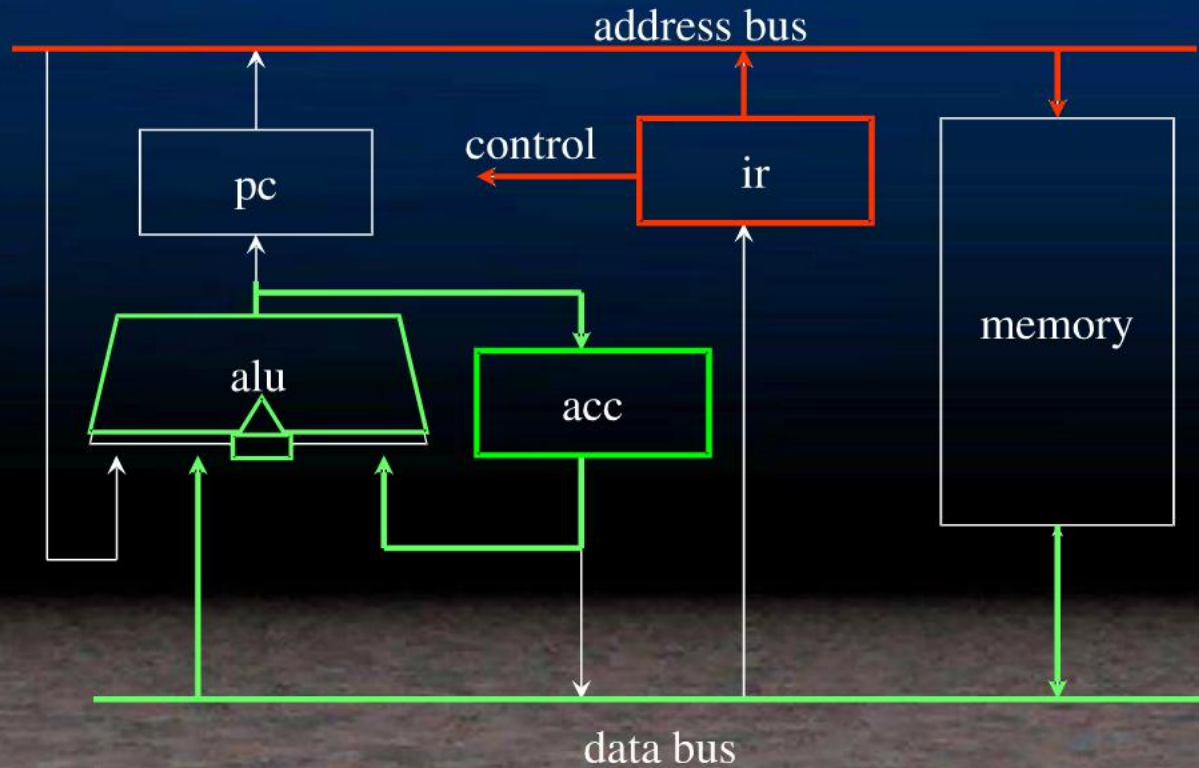
# Cycle1: 取指 & PC++

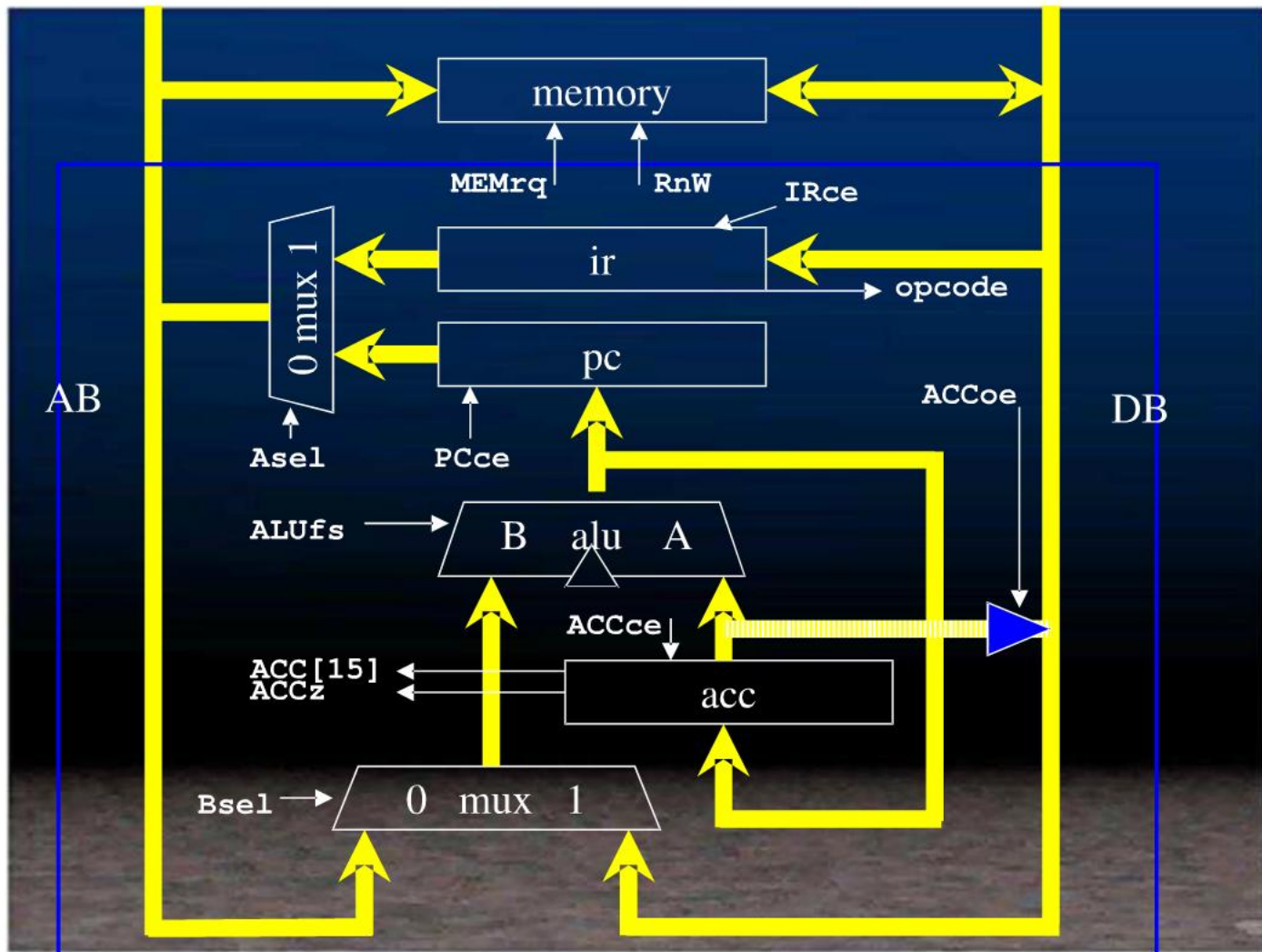


## Cycle2: 取操作数 + 执行

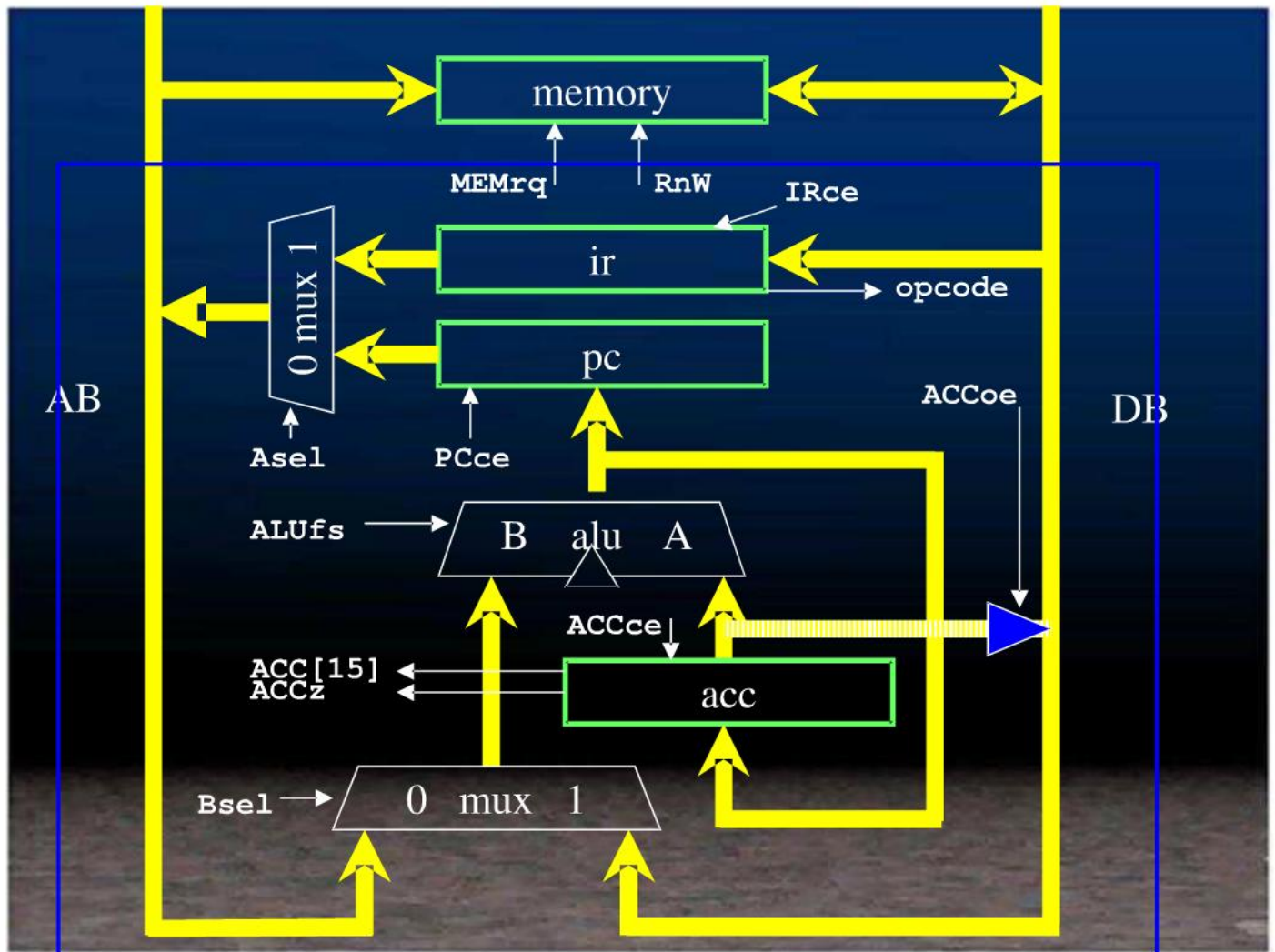


## Cycle2: 取操作数+ 执行

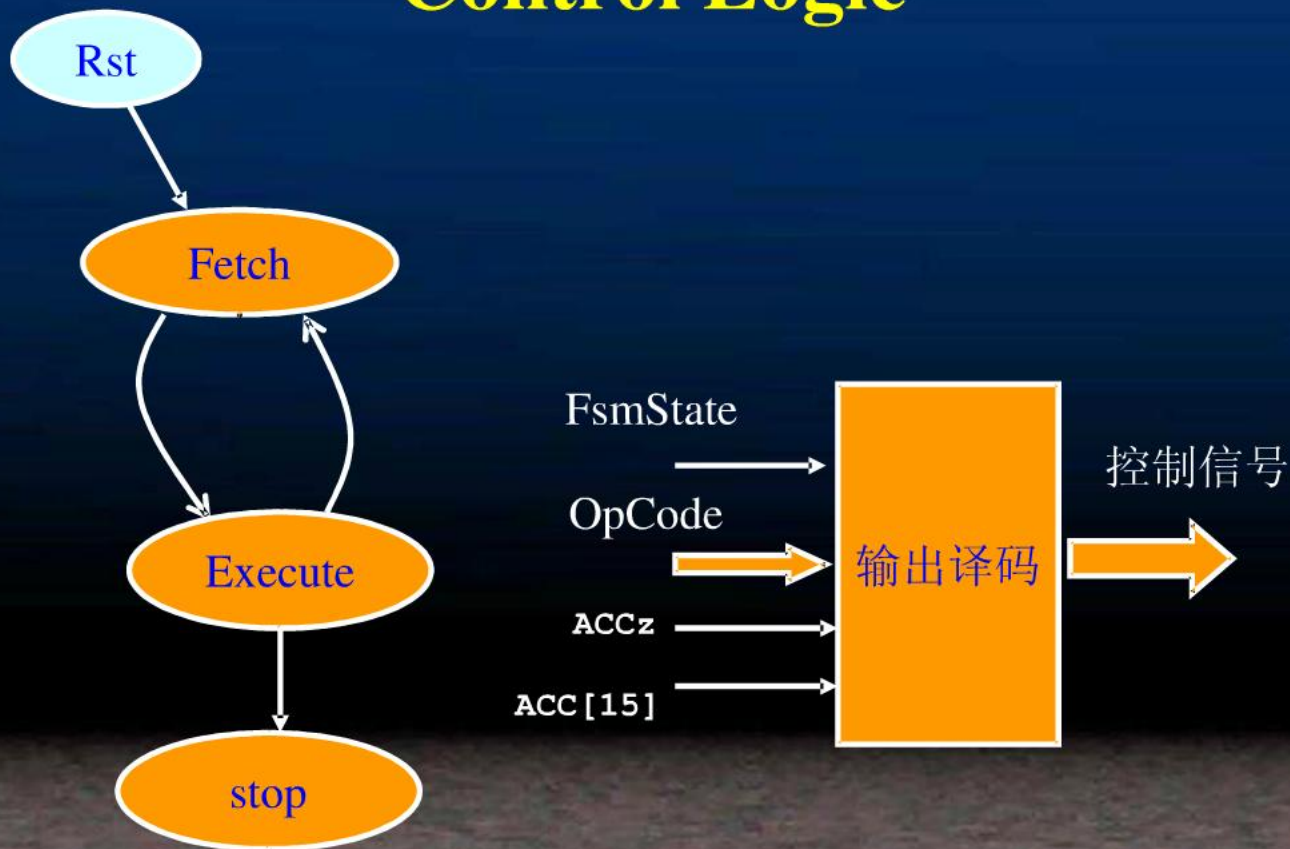








# Control Logic



## 需求再次升级:指令流控制

Instruction	Opcode	Semantics
LDA S	0000	$ACC := mem_{16}[S]$
STO S	0001	$mem_{16}[S] := ACC$
ADD S	0010	$ACC := ACC + mem_{16}[S]$
SUB S	0011	$ACC := ACC - mem_{16}[S]$
JMPS	0100	$PC := S$
JGES	0101	if $ACC \geq 0$ $PC := S$
JNES	0110	if $ACC \neq 0$ $PC := S$
STP	0111	stop

# MU0

- In this simple example, we'll look at the ISA and an obvious implementation of it.
- It's the MU0 -- that is, the Manchester University 0

# Components of MU0 (1)

- A **Program Counter (PC)** register to hold the address of the current instruction.  
(Visible to Programmer)
- One register - the **Accumulator (ACC)**.  
(Visible to Programmer).



## Components of MU0 (2)

- An **Arithmetic-Logic Unit (ALU)** for doing, eh, arithmetic and logic. (Invisible to Programmer).
- An **Instruction Register (IR)** for holding the current instruction. (Invisible to Programmer.)

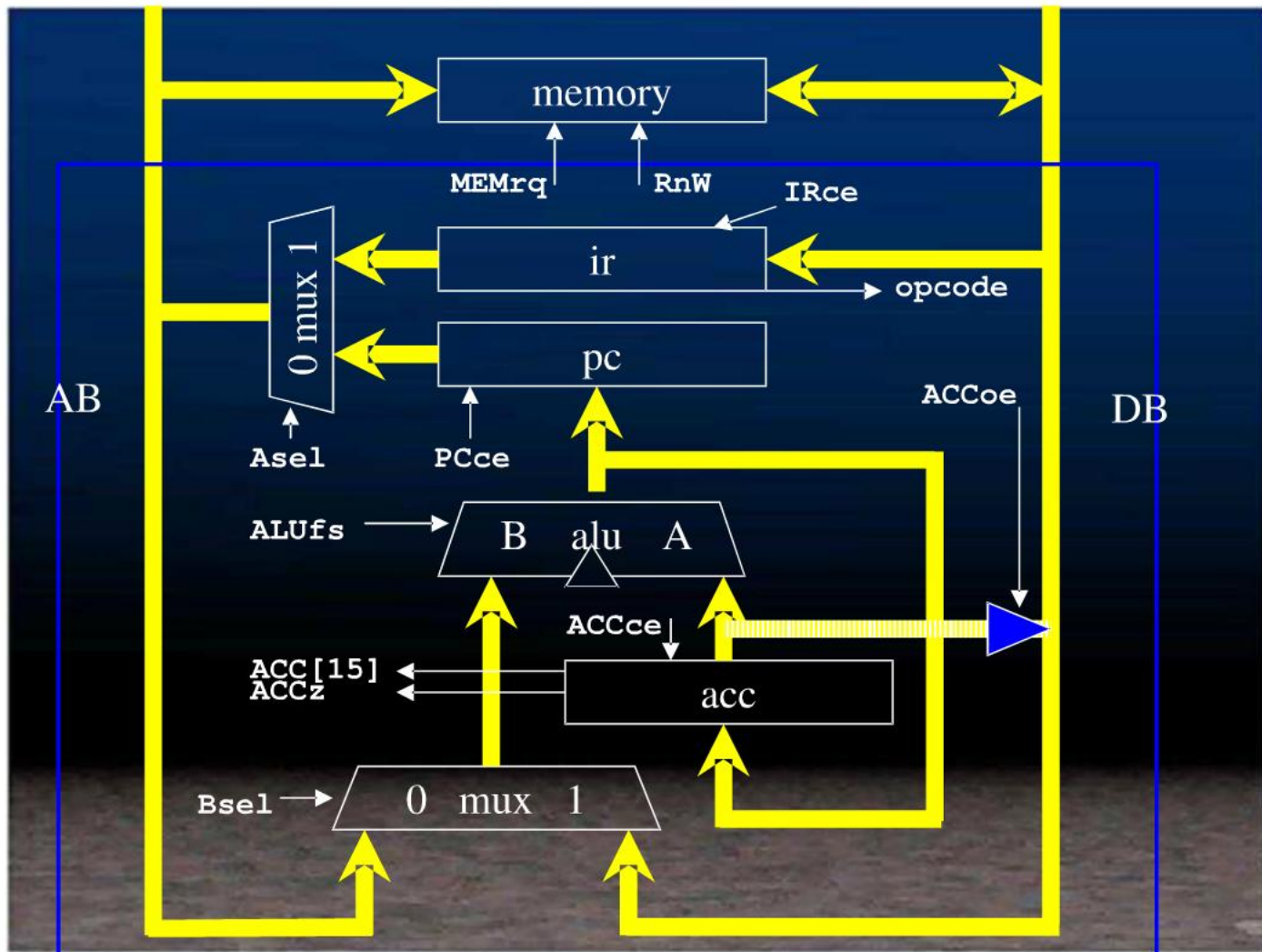
## Components of MU0 (3)

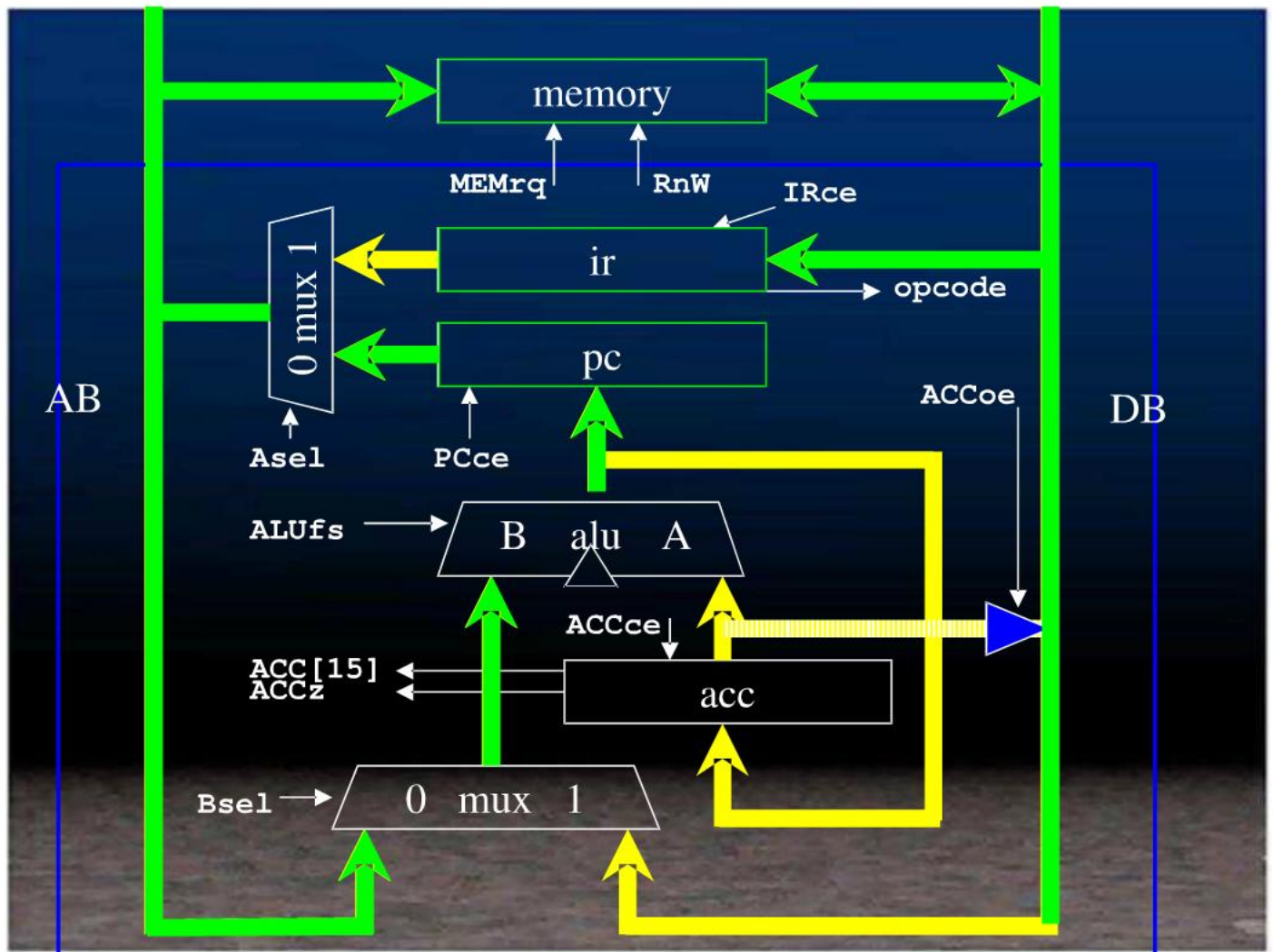
- **Instruction Decode Logic and Control Logic** to use the components of MU0 to give effect to the instructions.

# Features of MU0

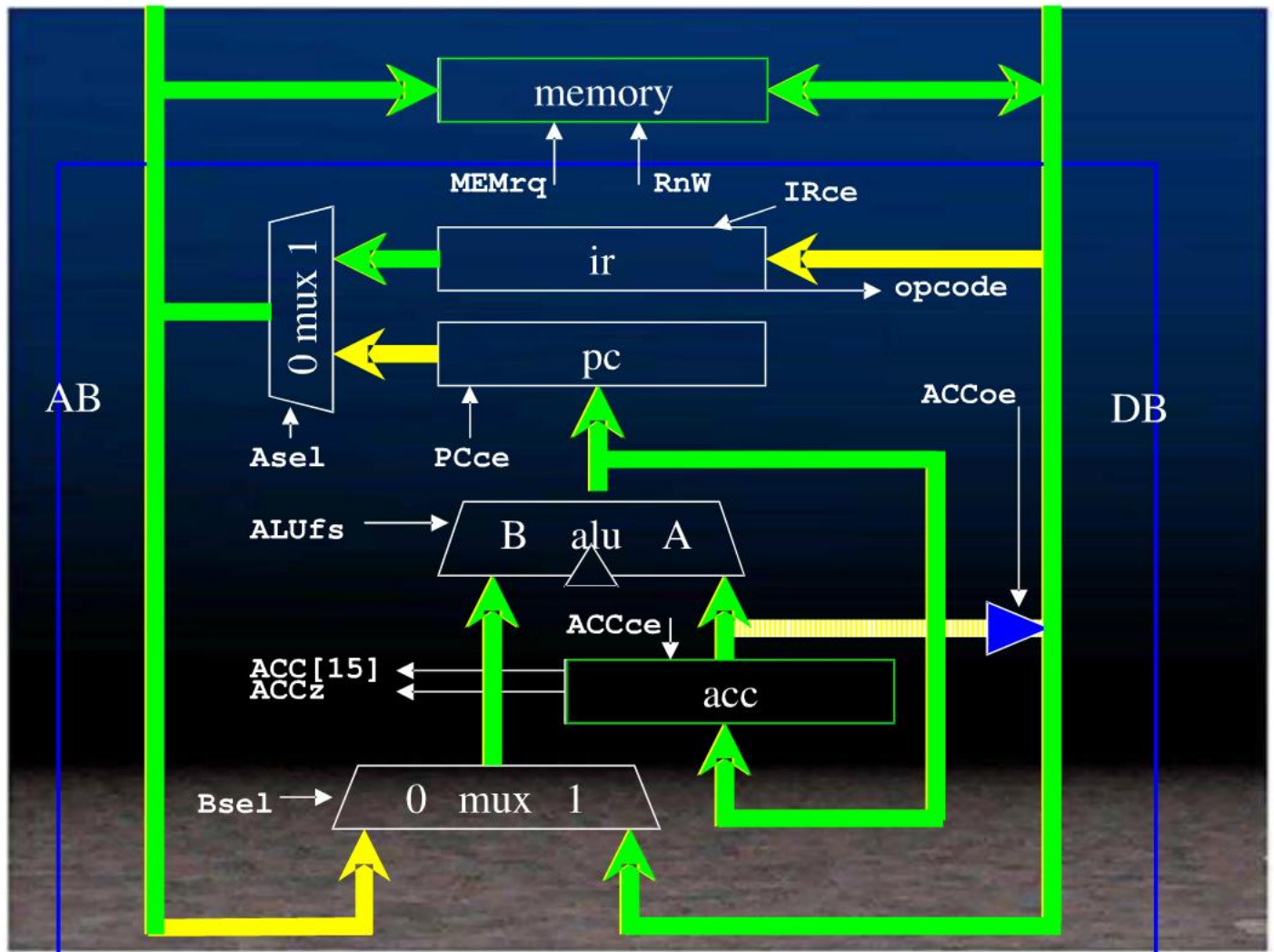
- 16 bit machine – ACC, PC, IR
- 2's Complement Arithmetic
- 12 bit addressing (see below)
- 16 bit instructions:

4 bits	12 bits
opcode	S

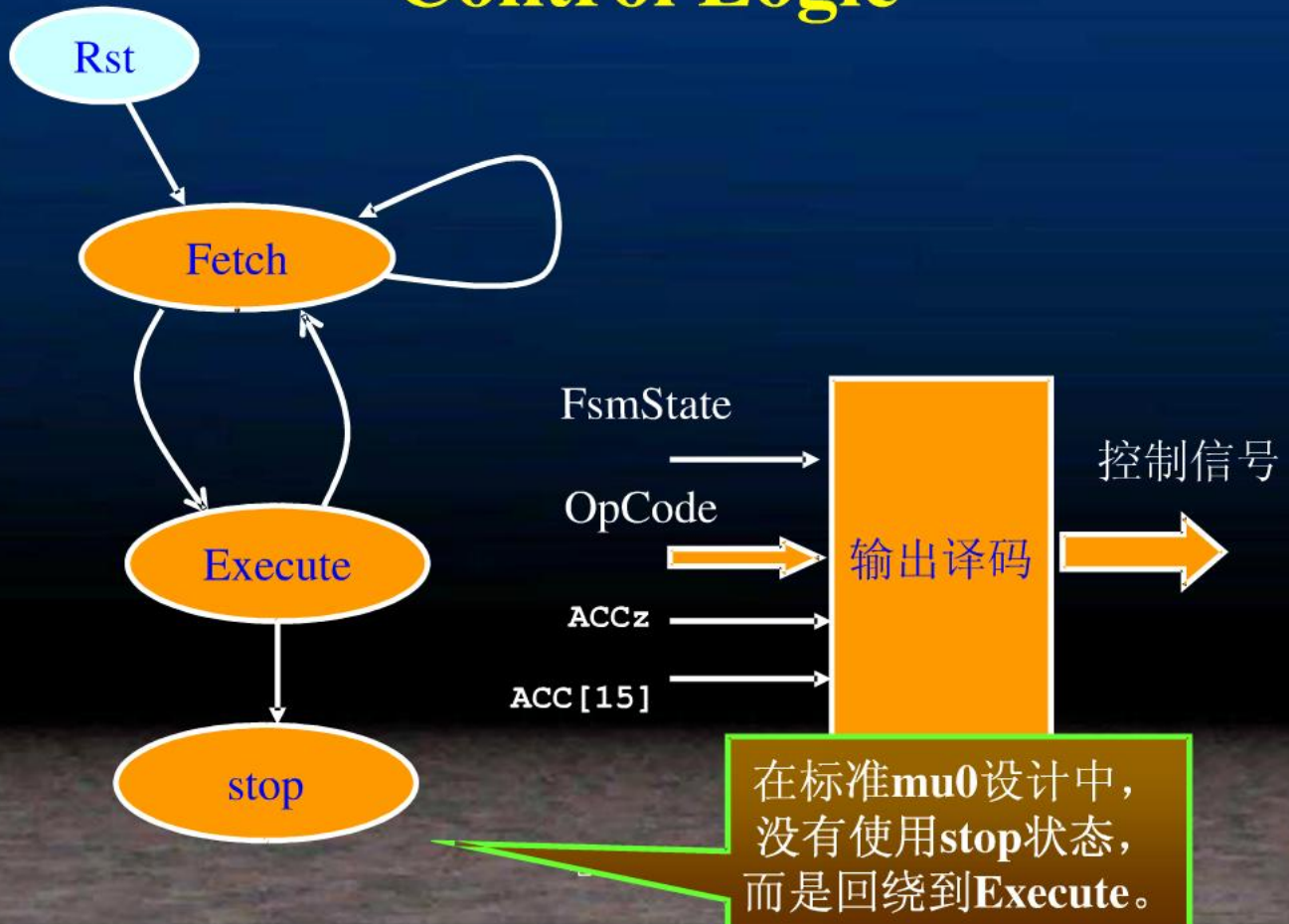








# Control Logic



# Control Logic Inputs

Inputs					
Instruction	Opcode	Reset	Ex/ft	ACCz	ACC15
Reset	xxxx	1	x	x	x
LDAS	0000	0	0	x	x
	0000	0	1	x	x
STO S	0001	0	0	x	x
	0001	0	1	x	x
ADD S	0010	0	0	x	x
	0010	0	1	x	x
SUB S	0011	0	0	x	x
	0011	0	1	x	x
JMP S	0100	0	x	x	x
JGE S	0101	0	x	x	0
	0101	0	x	x	1
JNE S	0110	0	x	0	x
	0110	0	x	1	x
STP	0111	0	x	x	x

# Control Logic Outputs

Outputs									
Aset	Bset	ACCce	PCce	IRce	ACCoe	ALUfs	MEMrq	RnW	Ex/ft
0	0	1	1	1	0	= 0	1	1	0
1	1	1	0	0	0	= B	1	1	1
0	0	0	1	1	0	B+1	1	1	0
1	x	0	0	0	1	x	1	0	1
0	0	0	1	1	0	B+1	1	1	0
1	1	1	0	0	0	A+B	1	1	1
0	0	0	1	1	0	B+1	1	1	0
1	1	1	0	0	0	A-B	1	1	1
0	0	0	1	1	0	B+1	1	1	0
1	0	0	1	1	0	B+1	1	1	0
1	0	0	1	1	0	B+1	1	1	0
0	0	0	1	1	0	B+1	1	1	0
1	0	0	1	1	0	B+1	1	1	0
0	0	0	1	1	0	B+1	1	1	0
1	0	0	1	1	0	B+1	1	1	0
0	0	0	1	1	0	B+1	1	1	0
1	x	0	0	0	0	x	0	1	0

# Cpu的性能提高策略

- 提高系统时钟频率：
  - 微码**ROM** → 硬连线逻辑；
  - 指令流水线。(流水线**mu0**需要将跳转指令的指令周期扩展成2个时钟)；
- 程序与数据总线分开：哈佛总线。
- 进一步提高**IPC**：超标量结构。
- 突破存储器带宽瓶颈：指令**Cache**和数据**Cache**。
- **DSP**中，引入硬件乘加器等以提高数据处理速度。



# 指令流水

指令1

取指 译码 执行

指令2

取指 译码 执行

指令3

取指 译码 执行

## 其他内容

- 可以参考有关资料。
- 对CPU设计感兴趣者可以参考：
  - 《ARM SoC体系结构》，北京航空航天大学出版社；
  - 《Modern Processor Design – Fundamentals of Superscalar Processors》，John Paul Shen。译本：《现代处理器设计----超标量处理器基础》，电子工业出版社。

# 小结

- 复杂数字系统设计：数据通路 + 控制逻辑；
- 需求分析 → 系统验证 → 模块划分及接口时序设计 → 状态机设计 → 代码编写 → 设计验证。

# 课程结束

谢谢！

电子科技大学