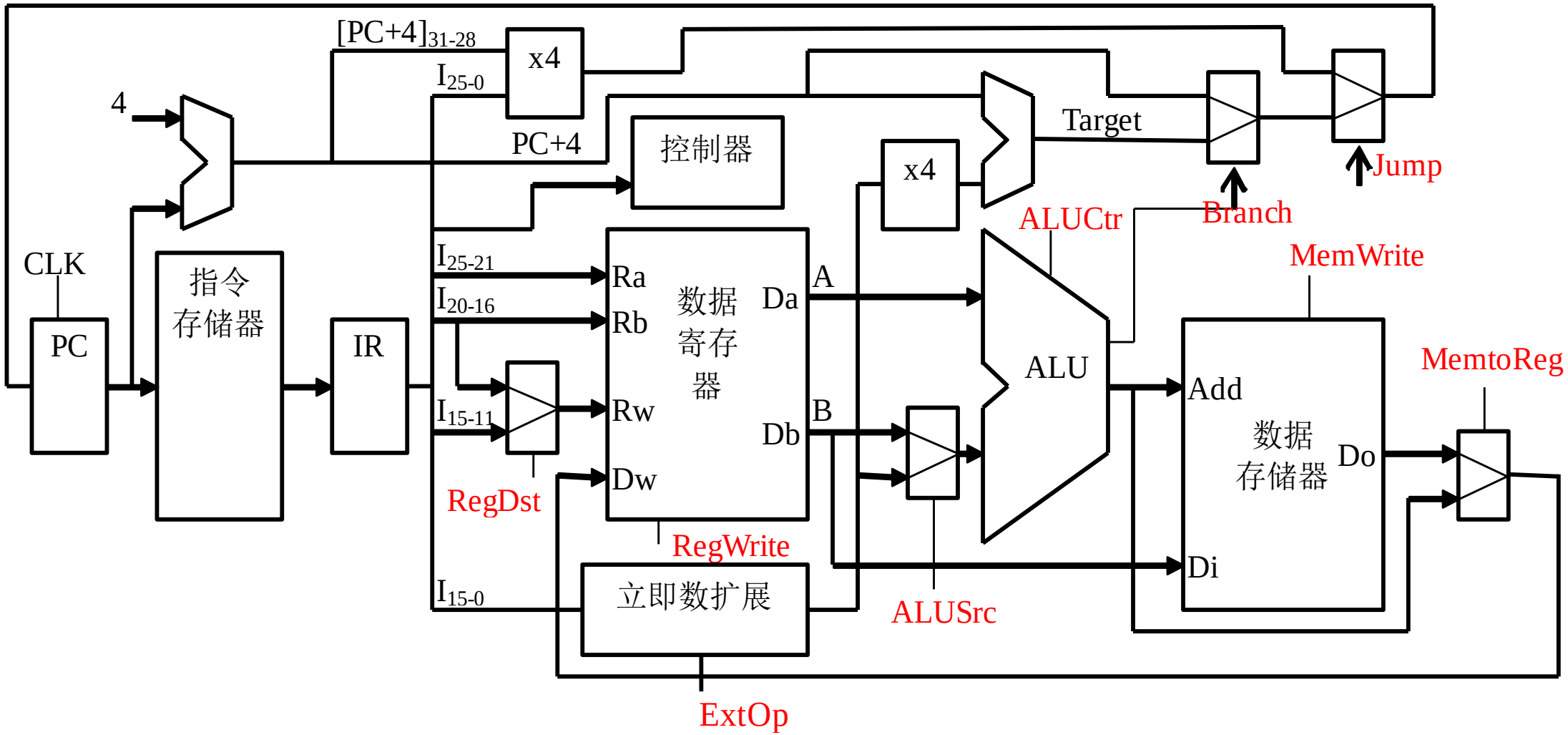
《计算机系统结构》课程直播

2020. 4.7

请将ZOOM名称改为“姓名”；

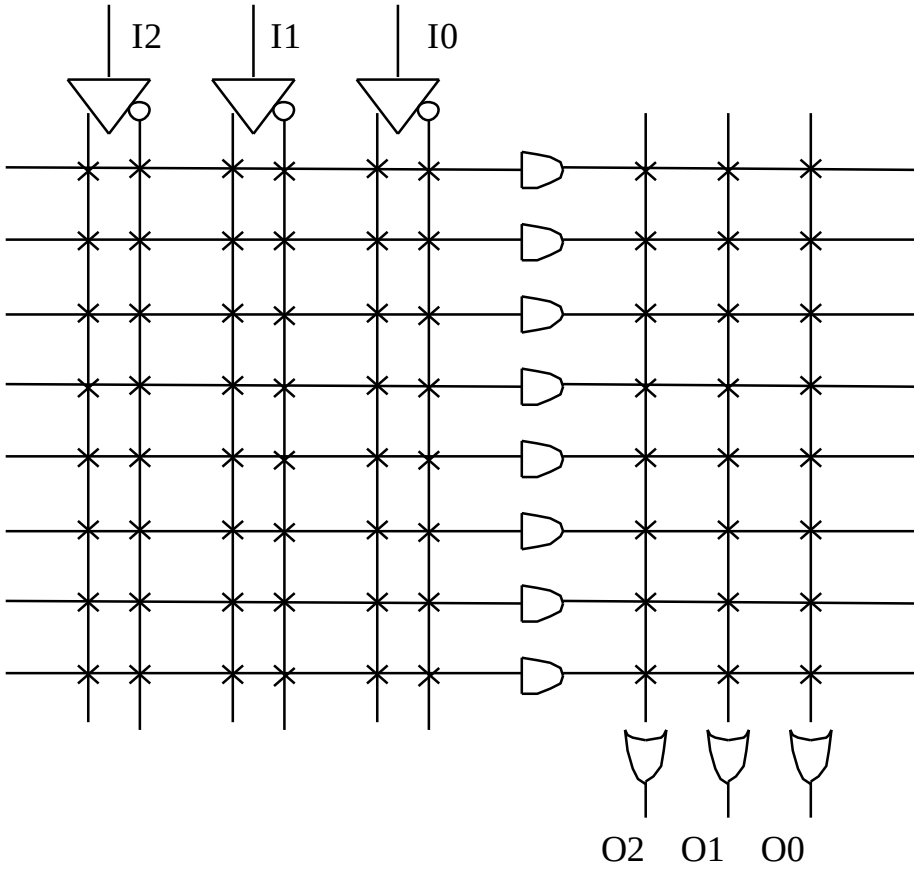
听不到声音请及时调试声音设备；签到将在课结束后继续

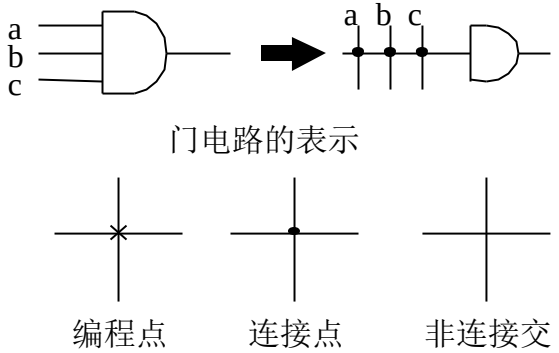
单周期处理器完整的数据通路



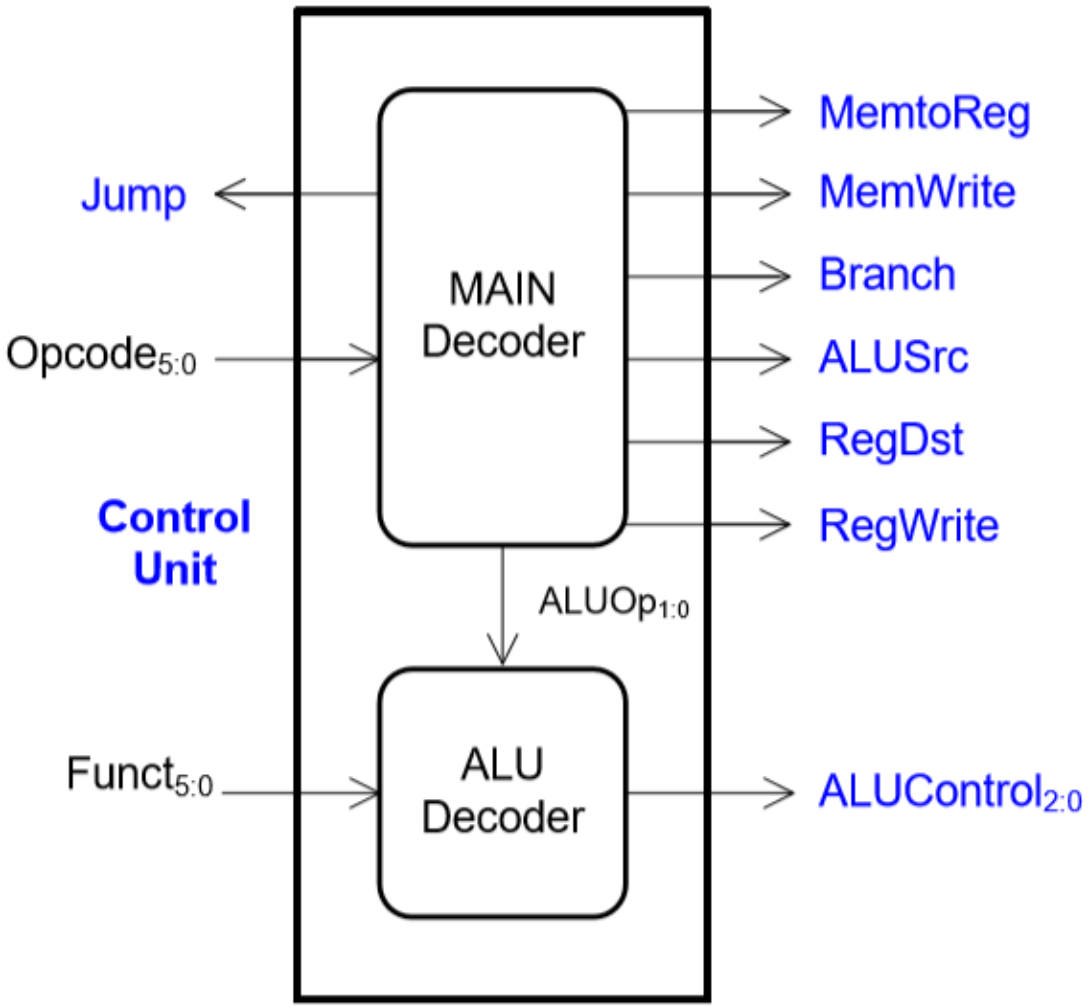
可编程逻辑阵列PLA

programmable logic array



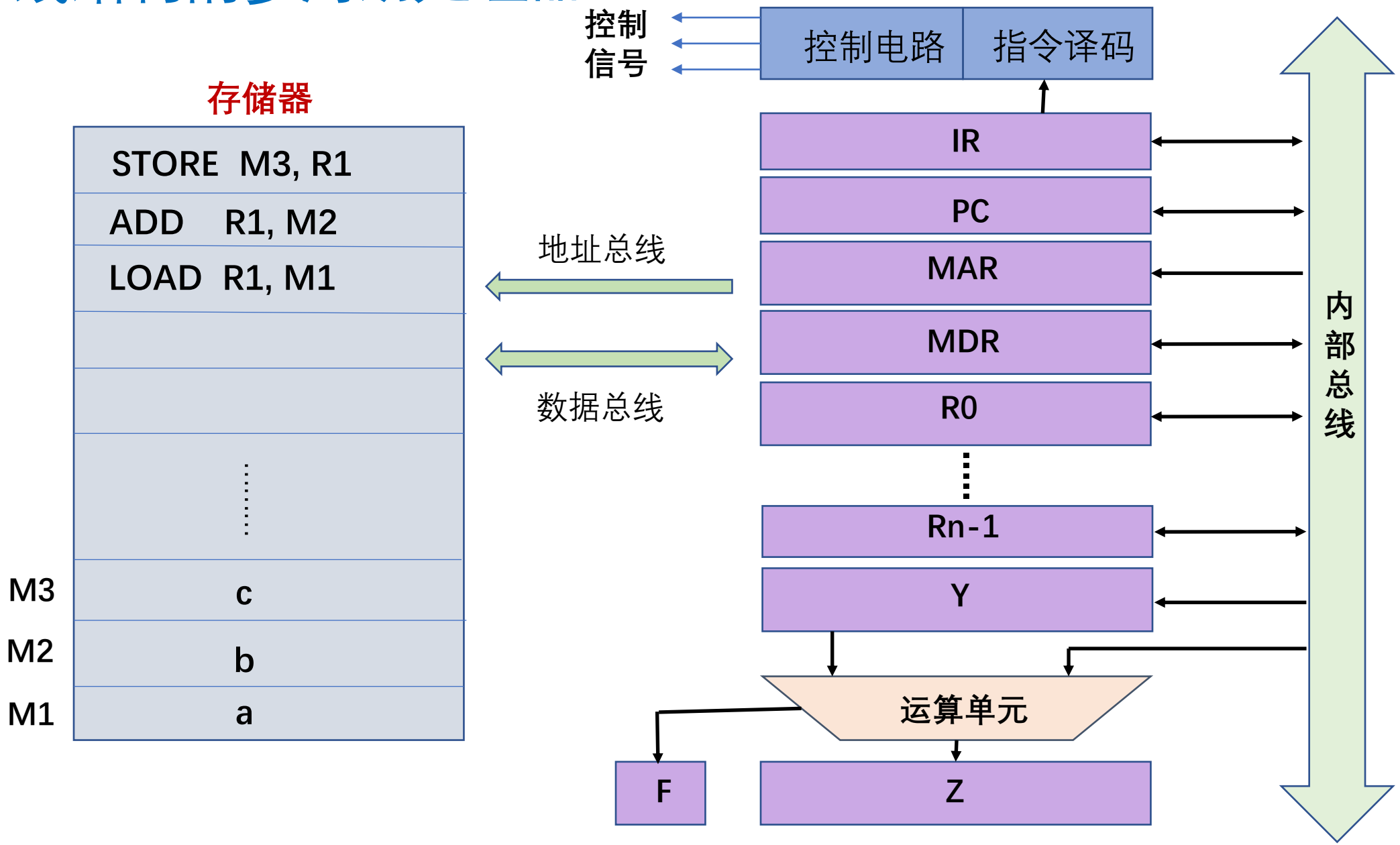


Control Logic



- The setting of the control lines is completely determined by the opcode fields of the instruction
- Focus on the design of the state machines to decode instructions and generate the sequence of control signals

单总线结构的多周期处理器



运算指令的执行过程

单总线结构：

- 如ADD R3, R1, R2

(1) PC→MAR

(2) PC+1→PC

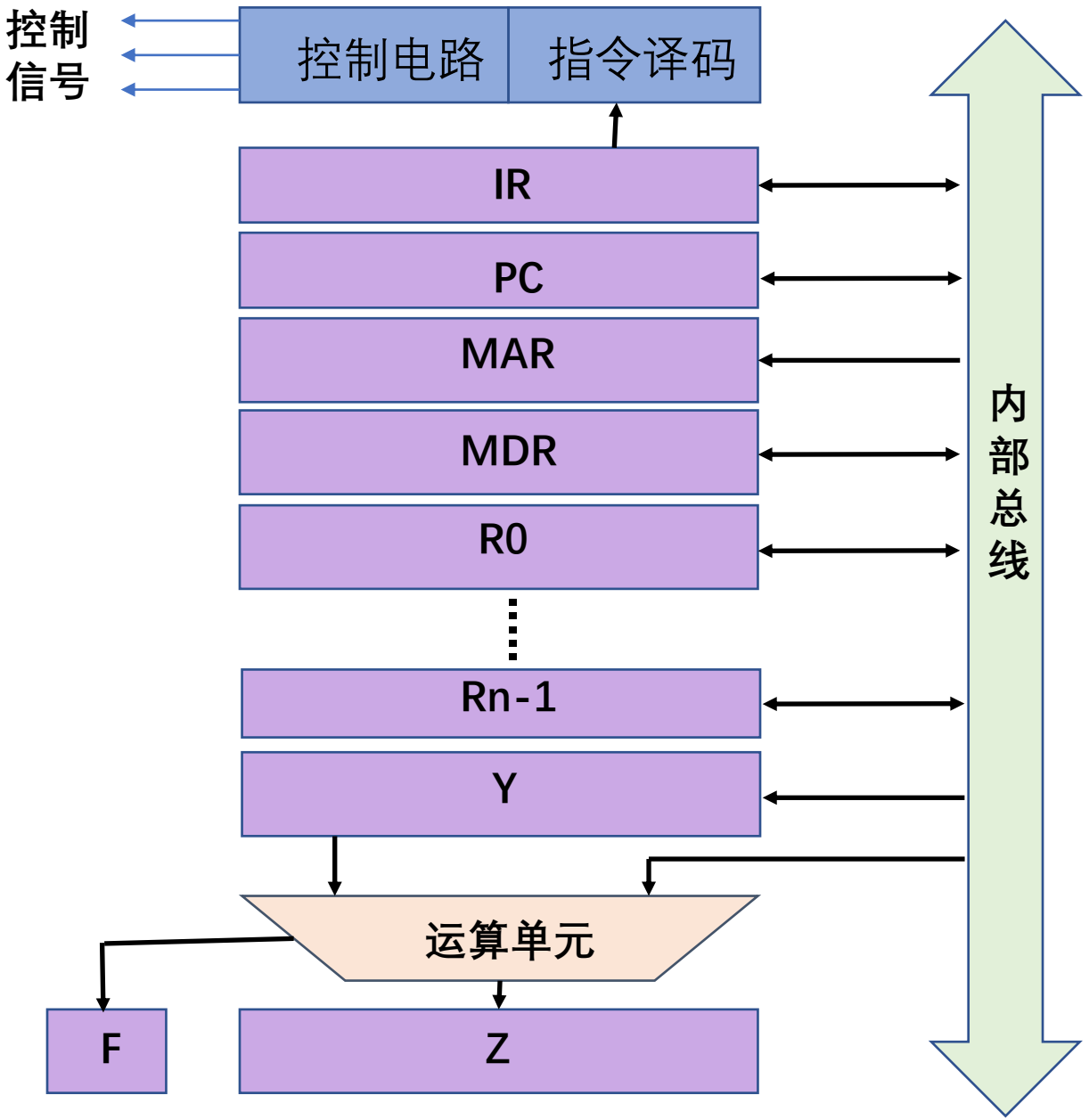
(3) DBUS→MDR

(4) MDR→IR

(5) R1→Y

(6) R2 + Y→Z

(7) Z→R3



访存指令的执行过程

单总线结构：

读操作：如LOAD R1,mem

(1) PC→MAR

(2) PC+1→PC

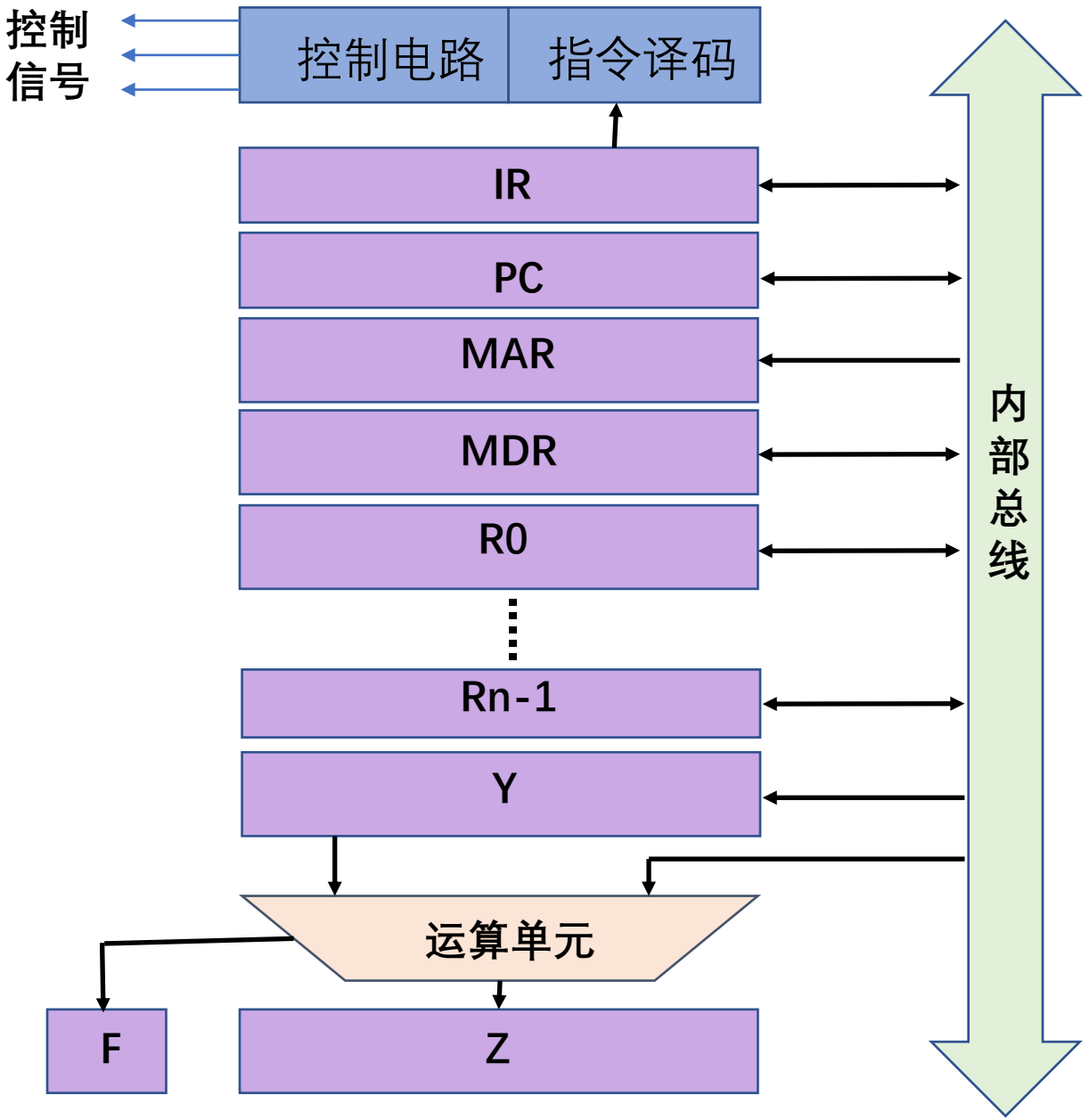
(3) DBUS→MDR

(4) MDR→IR

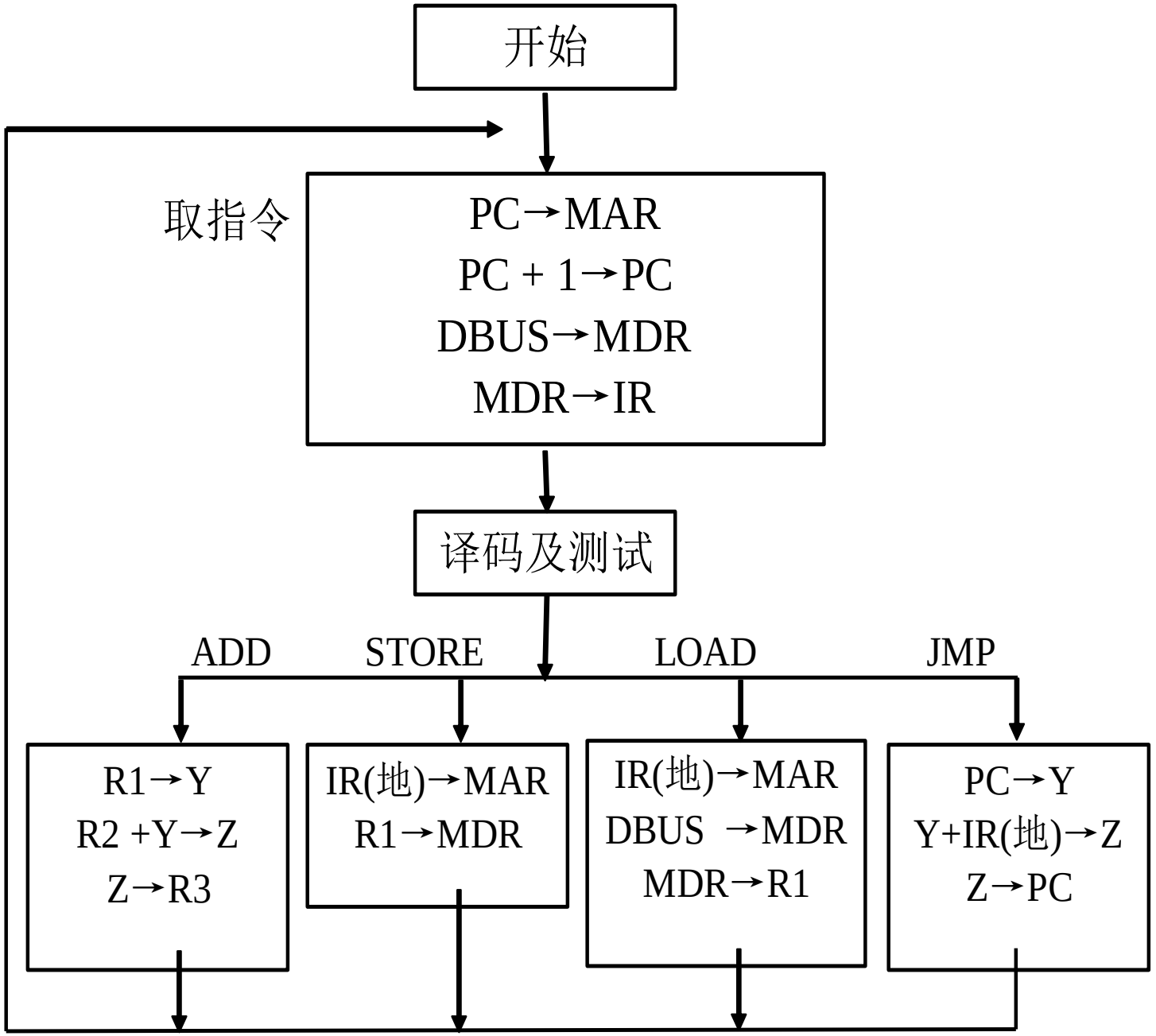
(5) IR(地址段)→MAR，读存储器

(6) DBUS→MDR

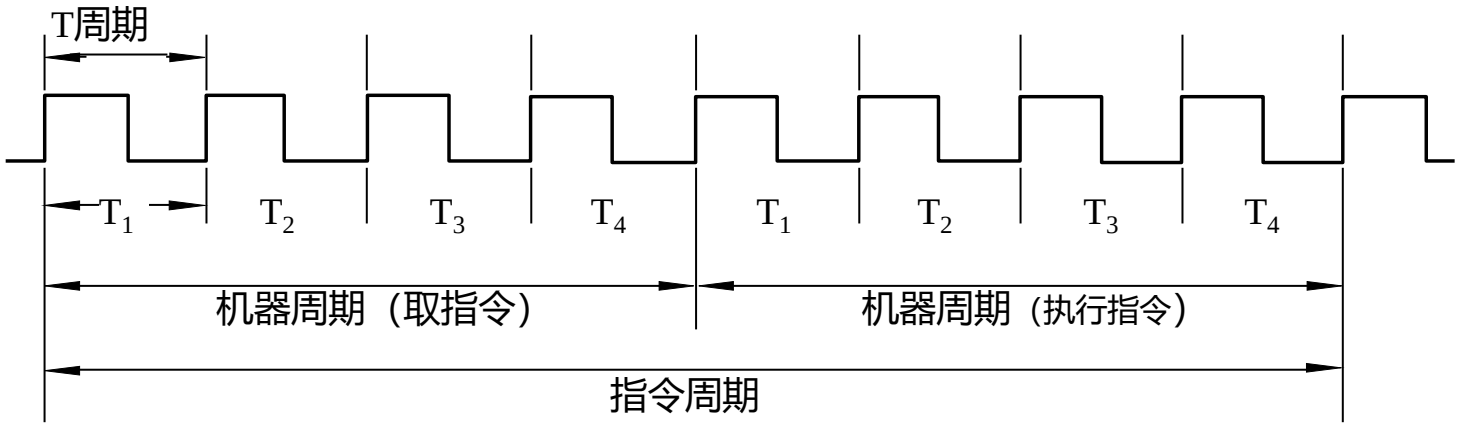
(7) MDR→R1



单总线结构处理器的指令执行**流程**



指令周期的基本概念



- 时钟周期：T,节拍脉冲
- 机器周期，CPU 周期：从内存读出一条指令的最短时间
- 指令周期：从内存取一条指令并执行该指令所用的时间。由若干个CPU周期组成。一个CPU周期又包含若干个时钟周期（节拍脉冲）

控制信号

如ADD R3, R1, R2

T1: PCout, MemRead, PC+1, MARin ;PC→MAR, PC+1→PC

T2: MDRout, IRin ;MDR→IR

addT3: R1out, Yin ;R1→Y

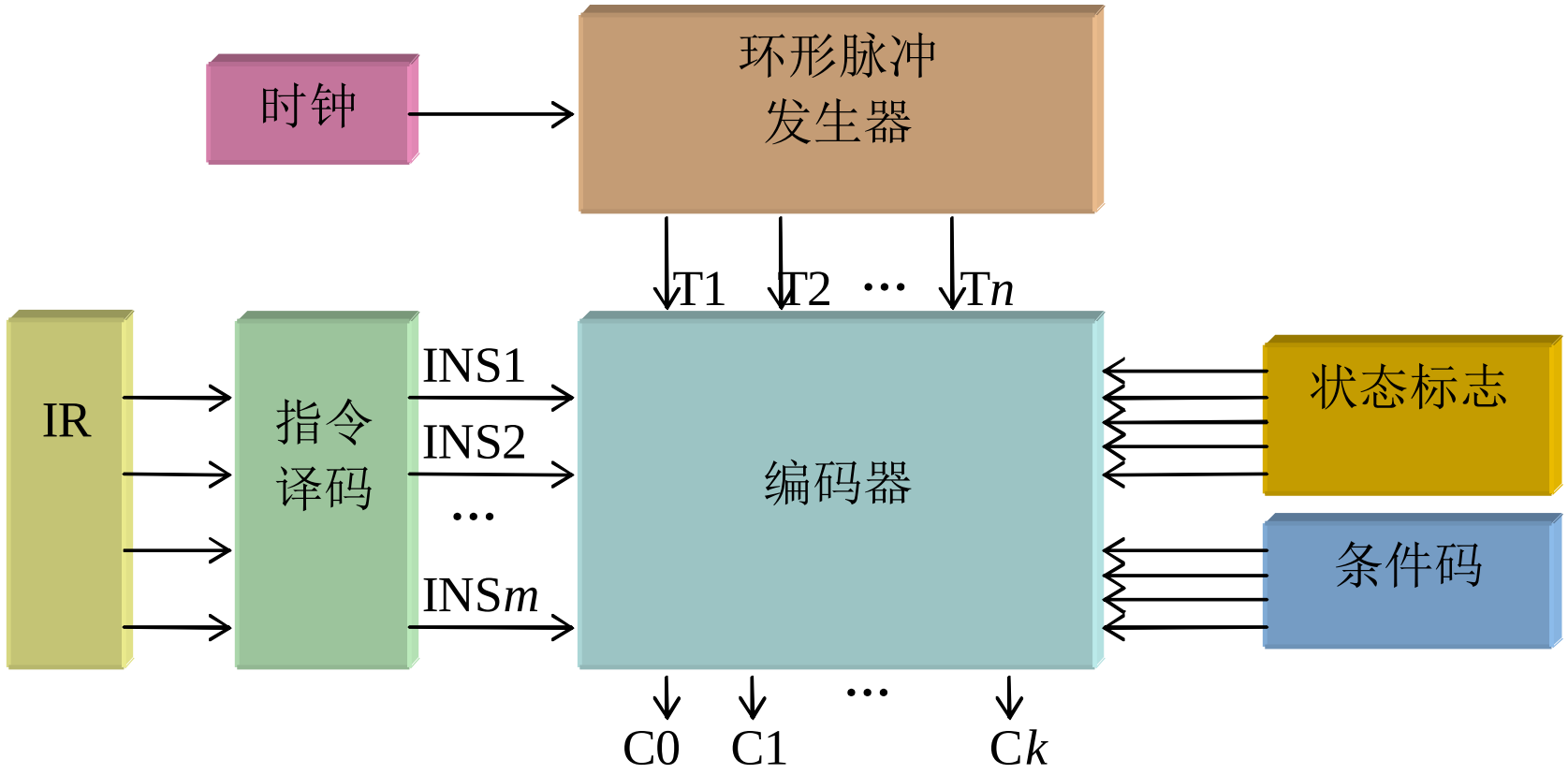
addT4: R2out, ADD, Zin ;R2+Y→Z

addT5: Zout, R3in ;Z→R3

小结

- 五阶段流水线处理器的实现
- 控制信号逐级传递
- 各段寄存器中存储的内容

硬连线控制器



Hard-wired controller

硬连线控制器(续)

条件码如算术运算的异常、内部中断等。

编码器电路的一般逻辑表达形式是:

$$Ci = T1*(INS1 + INS2 + \cdots) + T2*(INS1 + INS2 + \cdots) + \cdots$$

例：某计算机中包含2条指令：

ADD指令每个时钟周期内的控制信号为:

T1: PCout, MARin, PC+1, Read　　;PC→MAR, PC+1, read
T2: MDRout, IRin　　;MDR→IR
T3: R1out, Yin　　;R1→Y
T4: R2out, Zin, Add　　;R2+Y→Z
T5: Zout, R3in　　;Z→R3

JMP指令中各时钟周期的控制信号为:

T1: PCout, MARin, PC+1, Read　　; PC→MAR, PC+1, read
T2: MDRout, IRin　　;MDR→IR
T3: PCout, Yin　　;PC→Y
T4: IRout, Add, Zin　　;IR+Y→Z
T5: Zout, PCin　　;Z→PC

控制器的逻辑表达式如下

PC+1 = T1

PCin = T5*JMP

PCout = T1 + T3*JMP

Yin = T3*(ADD + JMP)

Add = T4*(ADD + JMP)

Zin = T4*(ADD + JMP)

Zout = T5*(ADD + JMP)

END = T5*(ADD + JMP)

...

硬布线控制器特点

- 组成的网络复杂;
- 无规则;
- 设计和调试困难;
- 不可改变指令系统和指令功能
- 适用于VLSI
- 速度快

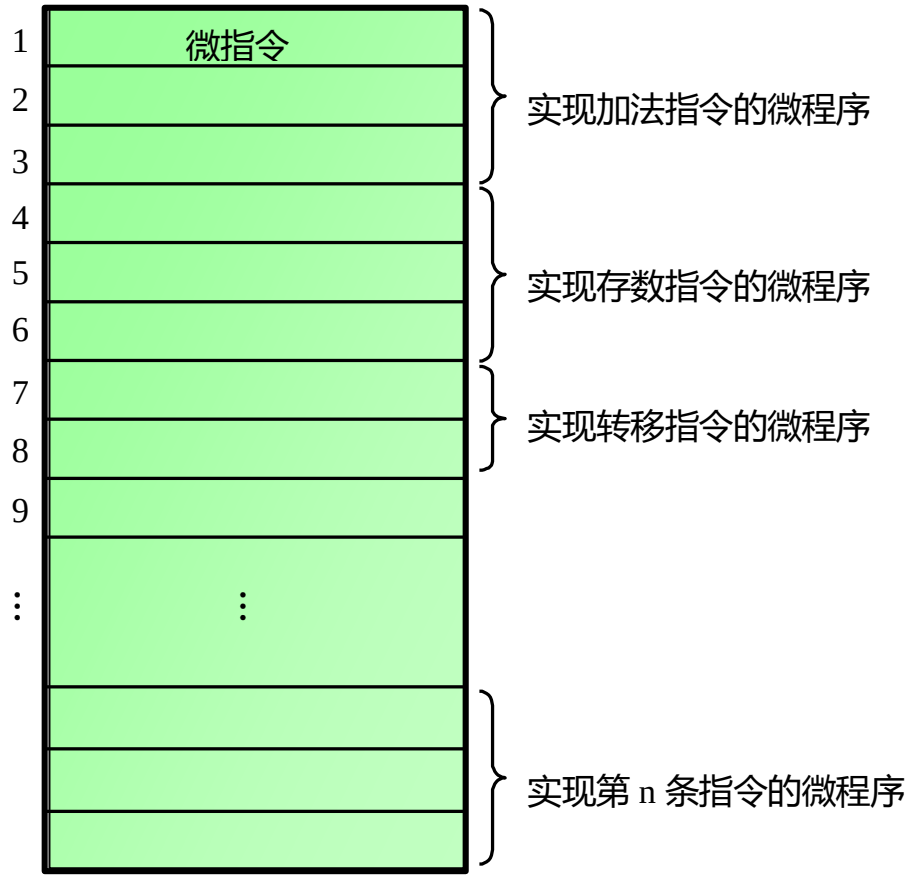
微程序控制概念microprogramming control

- 一条指令的处理包含许多微操作序列
- 将这些操作所需要的控制信号以多条微指令表示
- 执行一条微指令就给出一组微操作控制信号
- 执行一条指令也就是执行一段由多条微指令组成的微程序

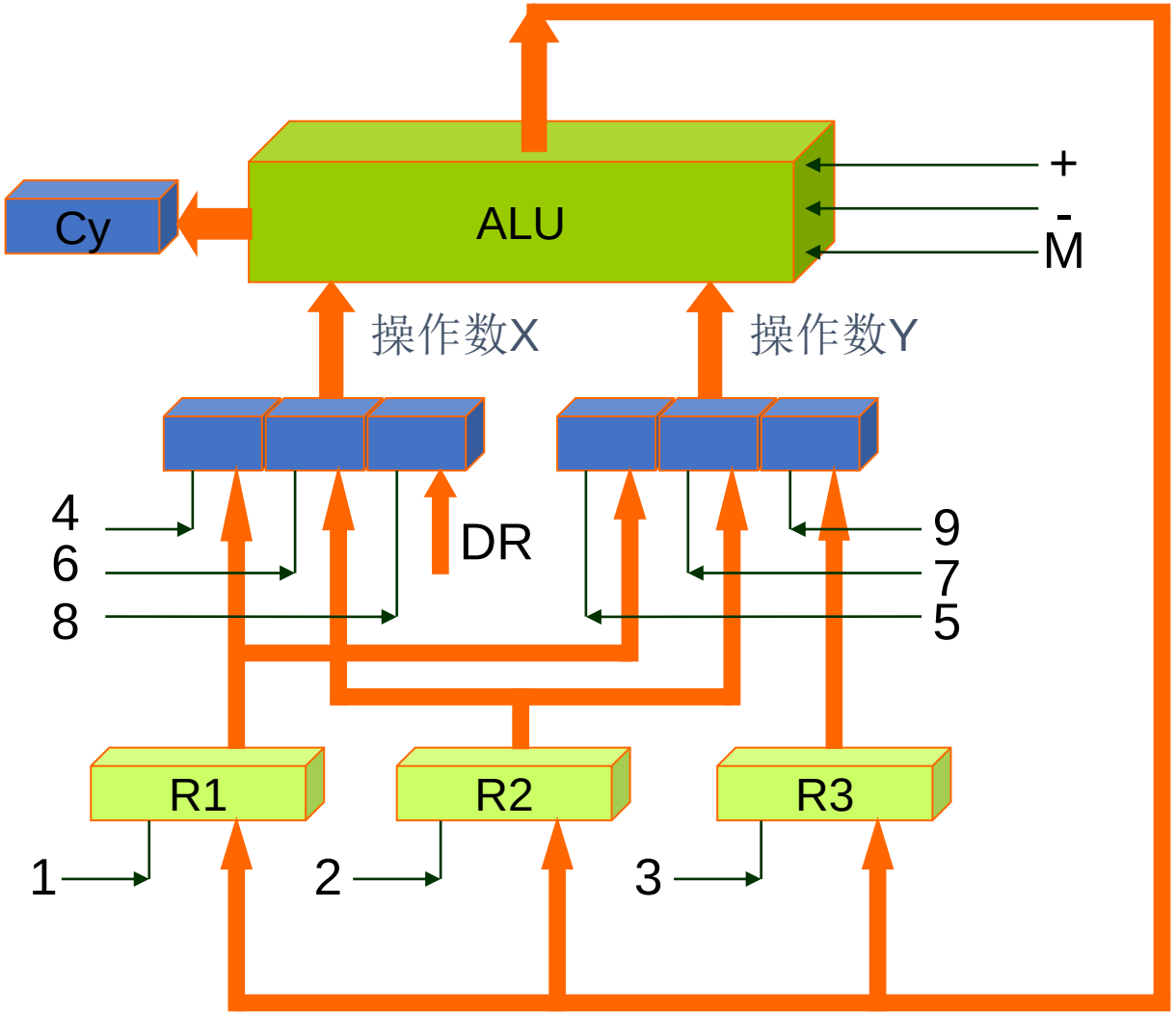
微程序控制概念

- 将指令系统功能实现所需的控制信号以微指令为单位存储。微指令中的每一位对应一根控制信号线
- 每条指令对应一段微程序
- 微程序由若干条微指令构成
- 机器执行指令时逐条取出微指令执行，使得相应部件执行规定的操作，执行完微程序，也就给出了该指令所需要的全部控制信号，从而完成一条指令的执行。

微指令与微程序

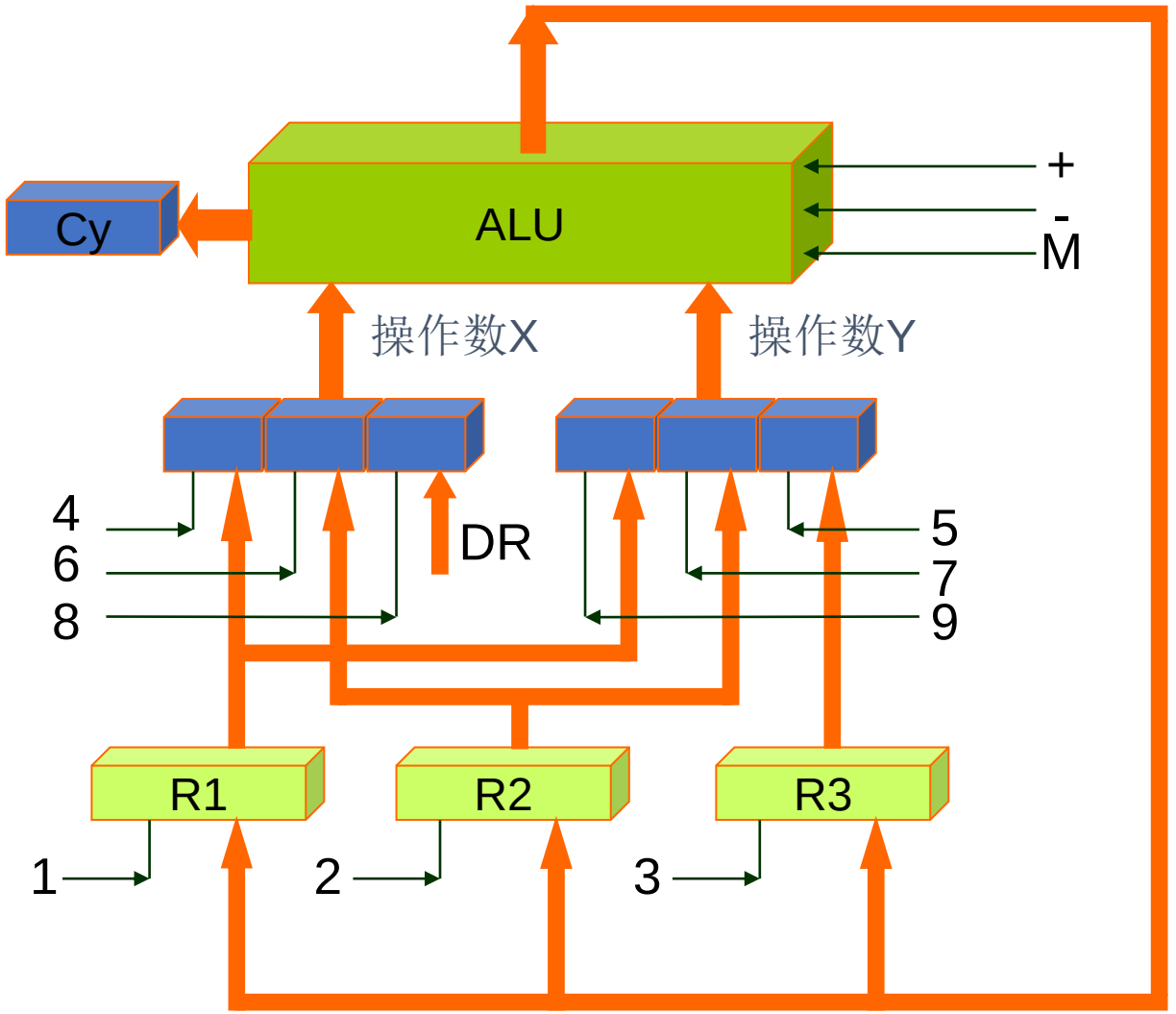

1
微指令
2
3
4
5
6
7
8
9
实现加法指令的微程序
实现存数指令的微程序
实现转移指令的微程序
实现第 n 条指令的微程序

简单运算器微命令



- 1：　LDR1
- 2：　LDR2
- 3：　LDR3
- 4：　R1→X
- 5：　R1→Y
- 6：　R2→X
- 7：　R2→Y
- 8：　DR→X
- 9：　R3→Y

简单运算器微命令

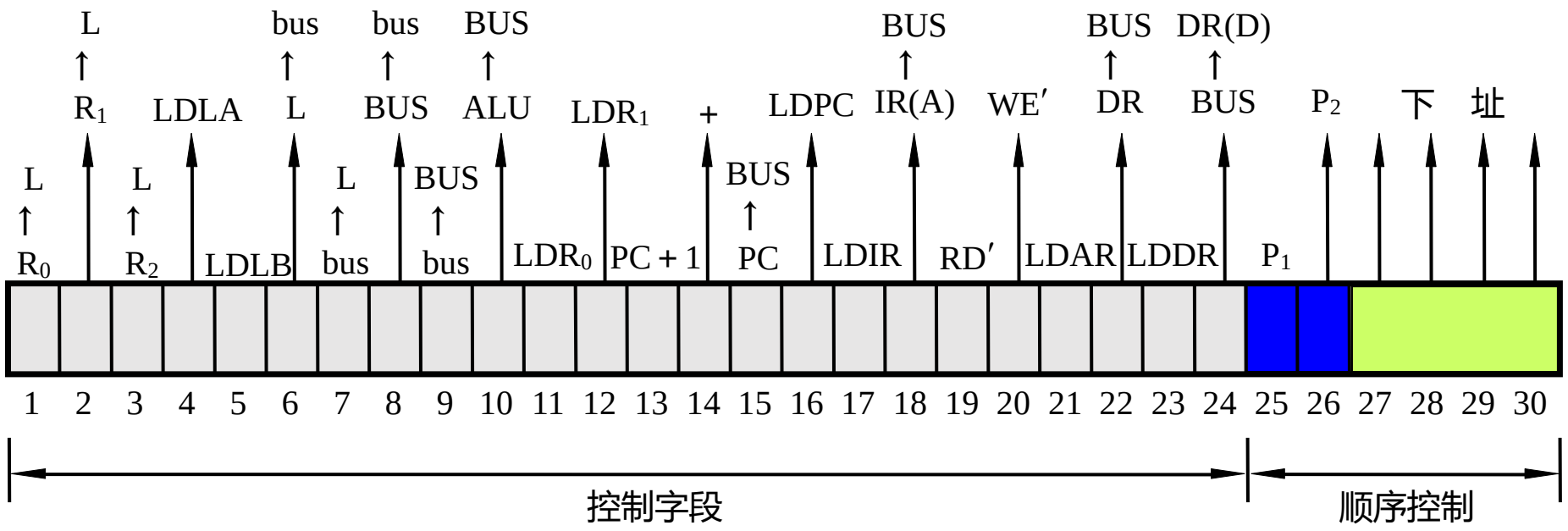


- 10：　+
- 11：　-
- 12：　M
- 13：　RD
- 14：　LDDR
- 15：　LDIR
- 16：　LDAR
- 17：　PC+1

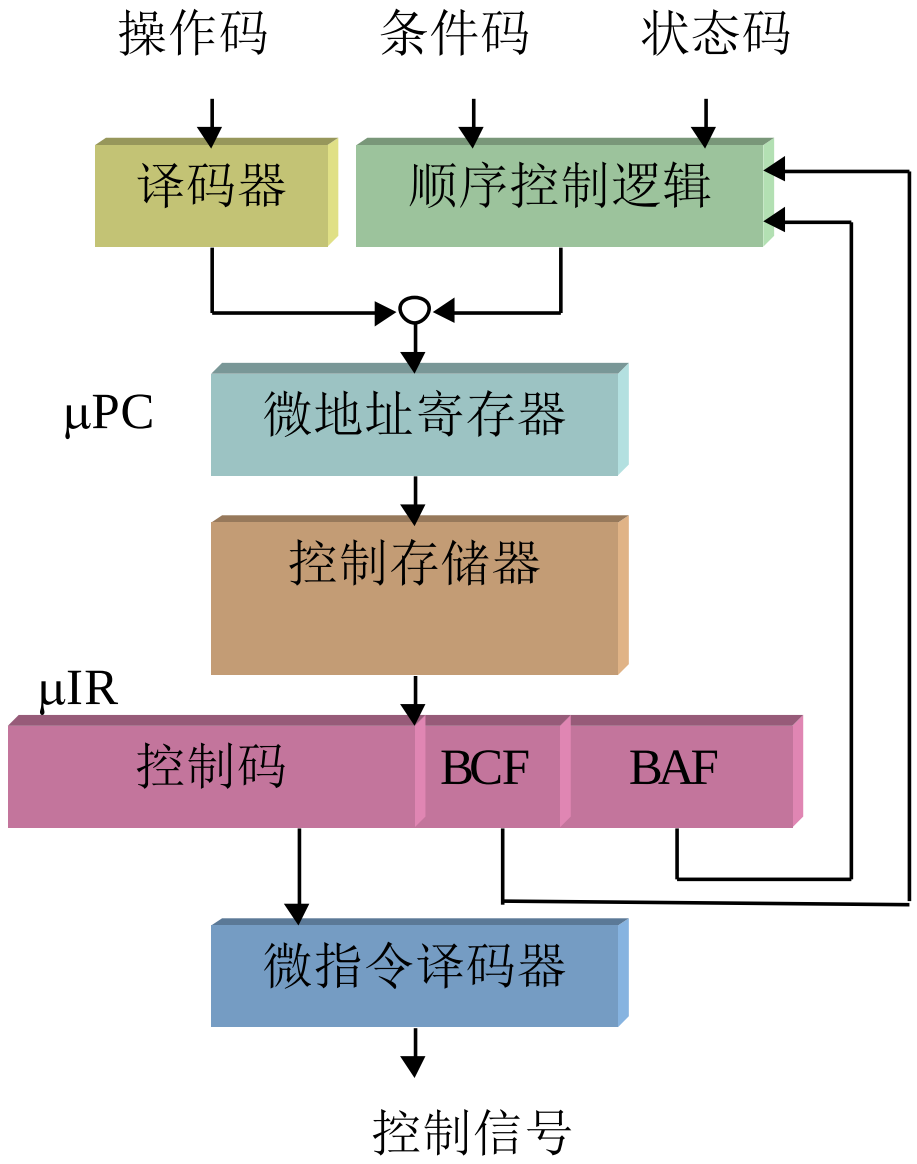
所有的微命令

1：LDR1
2：LDR2
3：LDR3
4：R1→X
5：R1→Y
6：R2→X
7：R2→Y
8：DR→X
9：R3→Y
10：+
11：-
12：M
13：RD
14：LDDR
15：LDIR
16：LDAR
17：PC+1

微指令基本格式


L
bus
bus
BUS
BUS
BUS
DR(D)
R1
LDLA
L
BUS
ALU
LDR1
+
LDPC
IR(A)
WE′
DR
BUS
P2
下
址
L
L
L
BUS
BUS
R0
R2
LDLB
bus
bus
LDR0
PC + 1
PC
LDIR
RD′
LDAR
LDDR
P1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
控制字段
顺序控制

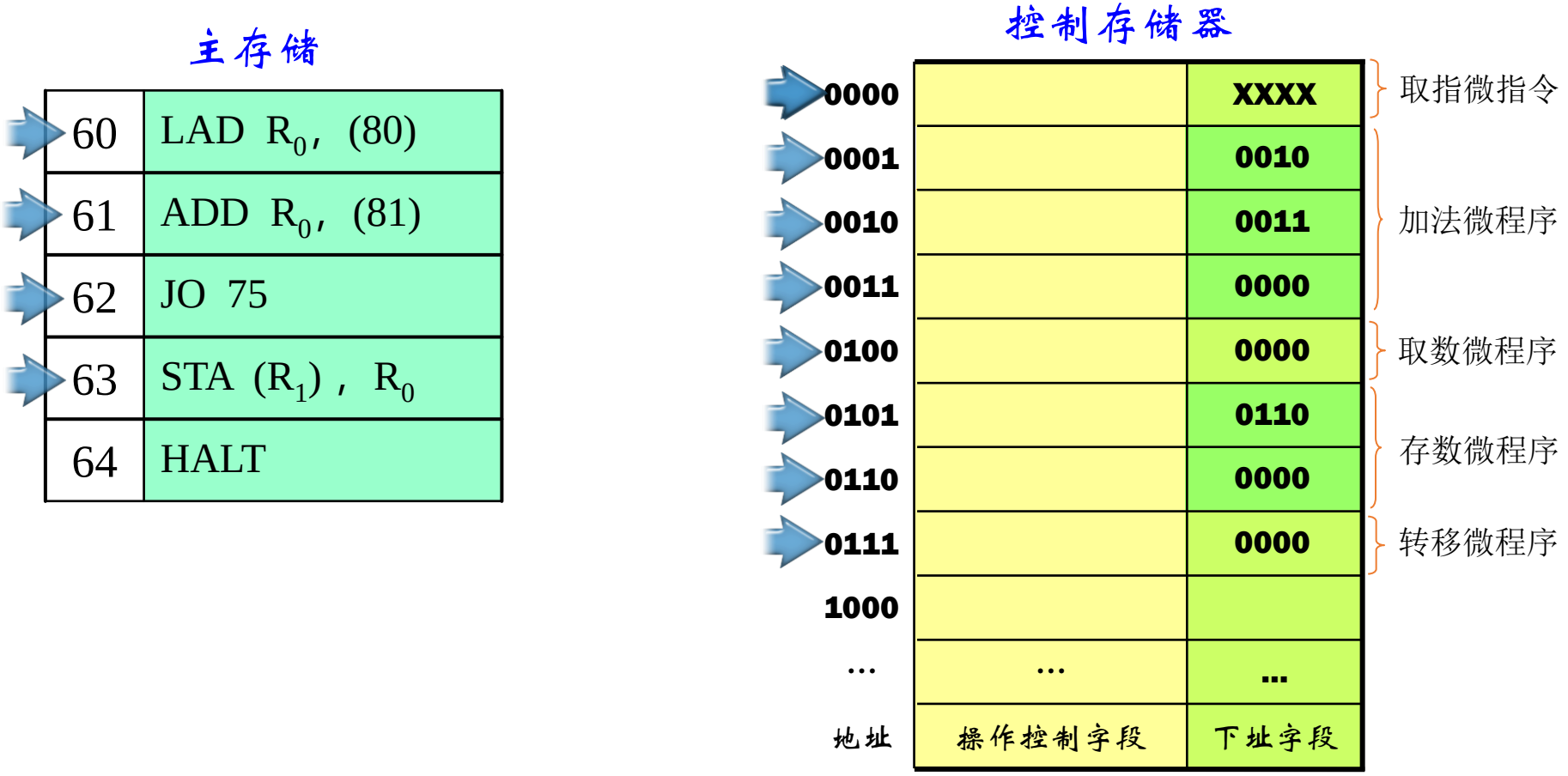
微程序控制器原理


操作码
条件码
状态码
译码器
顺序控制逻辑
μPC
微地址寄存器
控制存储器
μIR
控制码
BCF
BAF
微指令译码器
控制信号

执行指令微程序...

- 采用微程序控制的计算机的工作过程是执行微指令序列的过程。
- 微指令控制了取指令操作，
- 多条微指令实现了指令的功能。
- 而微指令中的微命令使执行部件完成微操作，计算机的工作过程是执行程序的过程，微观看，是执行指令的过程，再微观一点看，是执行部件进行微操作的过程

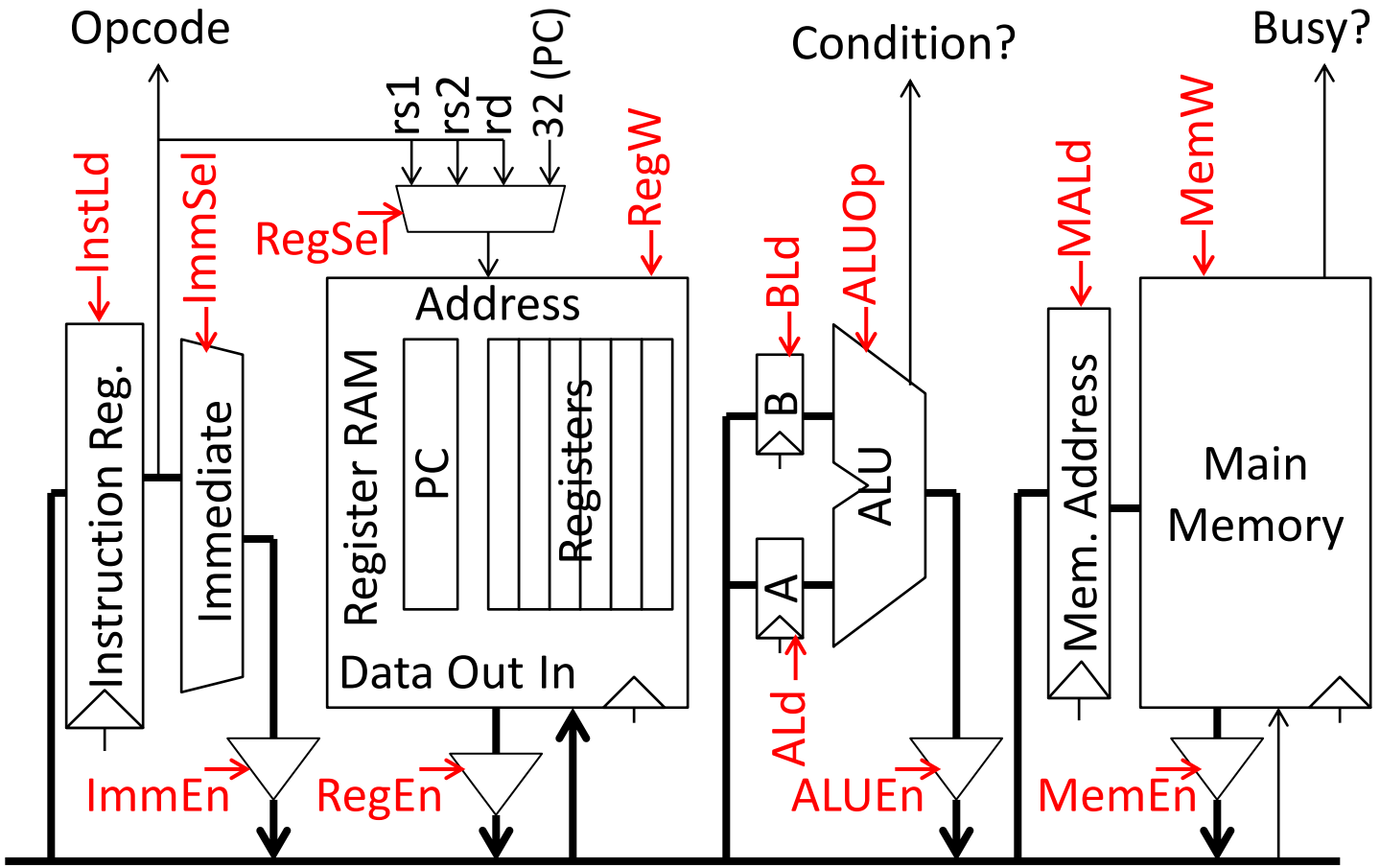
微程序存放示意图


主存储
60
LAD R0, (80)
61
ADD R0, (81)
62
JO 75
63
STA (R1), R0
64
HALT
控制存储器
0000
XXXX
0001
0010
0010
0011
0011
0000
0100
0000
0101
0110
0110
0000
0111
0000
1000
...
...
...
地址
操作控制字段
下址字段
取指微指令
加法微程序
取数微程序
存数微程序
转移微程序

为什么学习微程序控制

- 如何用一个简单处理器实现复杂的ISA
- CISC机器怎么发展起来的
  - CISC指令集仍然广泛使用 (x86, IBM360, PowerPC)
- 帮助我们理解技术驱动从CISC->RISC

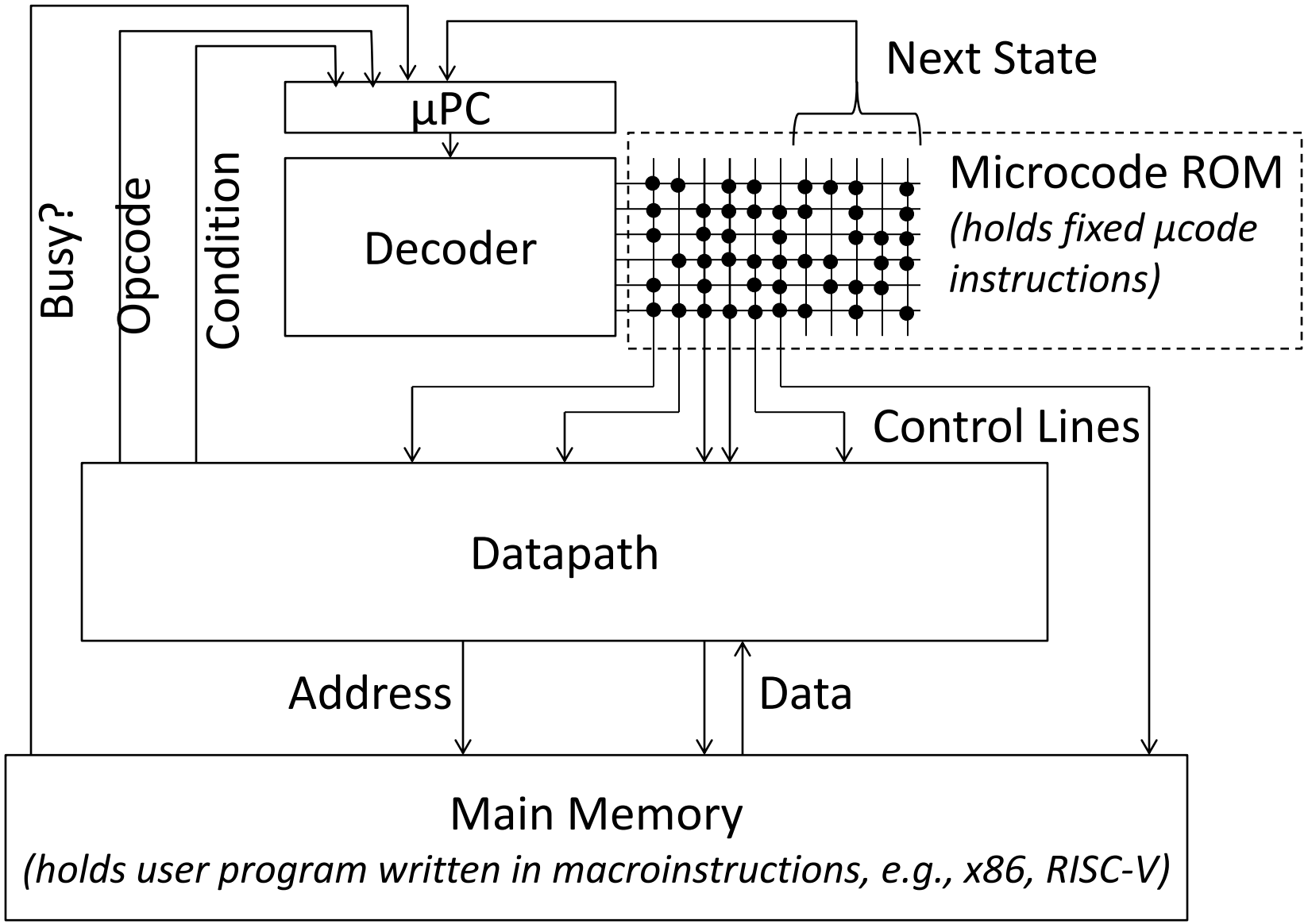
微程序控制RISC-V的单总线数据通路



微指令的寄存器传输级表示:

- MA:=PC means RegSel=PC; RegW=0; RegEn=1; MALd=1
- B:=Reg[rs2] means RegSel=rs2; RegW=0; RegEn=1; BLd=1
- Reg[rd]:=A+B means ALUop=Add; ALUEn=1; RegSel=rd; RegW=1

微程序控制 CPU


Next State
μPC
Busy?
Opcode
Condition
Decoder
Microcode ROM
(holds fixed μcode instructions)
Control Lines
Datapath
Address
Data
Main Memory
(holds user program written in macroinstructions, e.g., x86, RISC-V)

示例：实现一条复杂指令

Memory-memory add: M[rd] = M[rs1] + M[rs2]

Address	Data		
μPC	Control Lines	Next μPC	
MMA0	MA:=Reg[rs1]	next	
MMA1	A:=Mem		spin
MMA2	MA:=Reg[rs2]	next	
MMA3	B:=Mem		spin
MMA4	MA:=Reg[rd]	next	
MMA5	Mem:=ALUOp(A,B)		spin
MMA6			fetch

- 复杂指令的实现通常不需要修改数据通路，仅仅需要编写相应的微程序（可能会占用更多的控存）
- 采用硬布线控制器而不修改数据通路来实现这些指令是非常困难的

60到70年代微程序盛行

- ROM比DRAM要快的多
  - 逻辑器件（电子管）、主存（磁芯存储器）、ROM（二极管）
  - ROM和RAM速度之间的差异导致了额外的复杂指令
- 对于复杂的指令集，datapath和controller更便宜、更简单
  - 新的指令（例如 floating point）可以在不修改数据通路的情况下增加
  - 修改控制器的bug更容易
- 不同型号的机器实现ISA的兼容性更简单、成本更低

除了低档的或者性能最高机器，所有计算机都采用微程序控制

80年代初的微程序技术

- 微程序技术的进展孕育了更复杂的微程序控制的机器
  - 复杂指令集导致μcode需要子程序和调用堆栈
  - 需要修复控制程序中的bug与μROM 的只读属性冲突
- 超大规模集成电路技术出现，ROM和RAM速度的假设变得无效
  - 逻辑部件、存储部件（RAM, ROM）均采用MOS晶体管实现
  - 半导体RAM与ROM的存取速度相同
- 随着编译器技术的进步复杂指令变得不再那么重要
- 随着微结构技术的进步（pipeling, caches and buffers）,使多周期执行reg-reg指令失去了吸引力

80年初：微程序控制机器分析

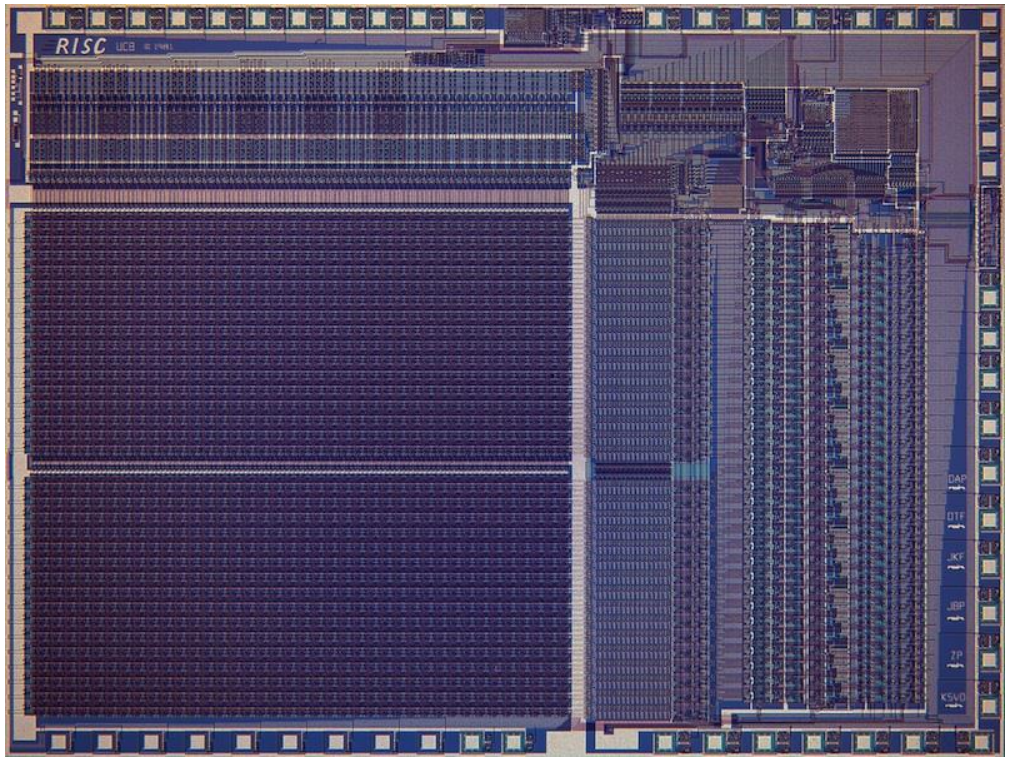
- 用高级语言编程成为主流
  - 关键问题：编译器会生成什么指令?
- IBM的John Cocke团队
  - 为小型计算机801（ECL Server）开发了更简单的 ISA 和编译器
  - 移植到IBM370，仅使用IBM 370的简单的寄存器-寄存器及load/store指令
  - 发现：与原IBM 370相比，性能提高3X
- 80年代初，Emer和Clark （DEC）发现
  - VAX 11/180 CPI = 10!
  - 虽然声称是1MIPS的机器，实际测试其性能是0.5MIPS
  - VAX ISA 的 20%指令 （占用了60%的微码）仅占用了 0.2%的执行时间
- VAX8800
  - 控制存储: 16K*147b RAM, Unified Cache: 64K*8b RAM
  - 微程序控制存储是cache容量的4.5x

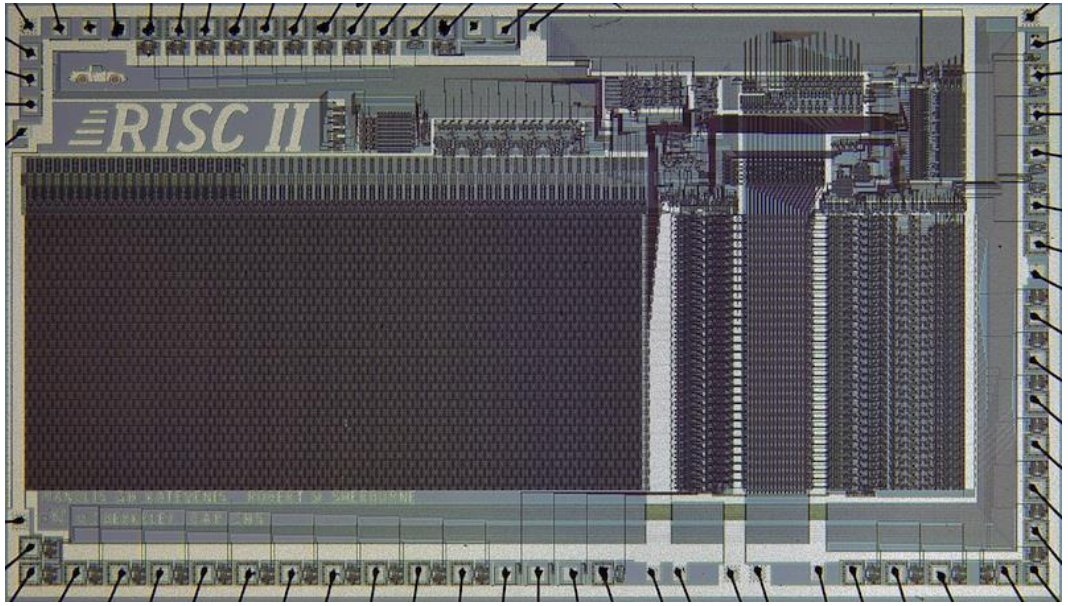
From CISC to RISC

- 使用快速RAM构建用户最近要执行的指令的指令缓存，而不是固定的硬件微程序
  - 指令缓存中的内容随着程序执行不断更新，提高访存速度
  - 使用简单的ISA，以有效实现硬布线的流水线方式执行
  - 大多数编译器生成的代码只使用了一部分常用的CISC指令
  - 简单的指令格式使得流水线高效实现成为可能
- 芯片集成度的提高带来的机遇
  - 80年代初，单芯片上已经可以集成32-bit的数据通路加上小的cache
  - 大多数情况下没有芯片间的通信，使得性能更好

Berkeley RISC Chips

RISC-I (1982) Contains 44,420 transistors, fabbed in 5 µm NMOS, with a die area of 77 mm^2, ran at 1 MHz. This chip is probably the first VLSI RISC.





RISC-II (1983) contains 40,760 transistors, was fabbed in 3 µm NMOS, ran at 3 MHz, and the size is 60 mm^2.

Microprogramming is far from extinct

- 80年代微程序控制起到了关键作用
  - DEC uVAX, Motorola 68K series, Intel 286/386

- 现代微处理器中微程序控制扮演辅助的角色
  - e.g., AMD Bulldozer, Intel Ivy Bridge, Intel Atom, IBM PowerPC, …
  - 大多数指令采用硬布线逻辑控制
  - 不常用的指令或者复杂的指令采用微程序控制

- 芯片bug的修复（打补丁） 例如Intel处理器在bootup阶段可装载微代码方式的patches
  - 英特尔不得不重新启用微代码工具，并寻找原来的微代码工程师来修补熔毁/幽灵安全漏洞

下周二

- 流水线处理器
- 精确中断的概念和实现
- 请预习5.6 、 5.7

再见