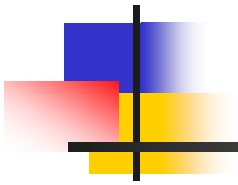


计算机组织与结构（下）
Computer Organization and Architecture

信息科学与工程学院

2019年2月28日

课程安排

- 理论课2学时。
- 课程设计24学时。地点：信息学院实验中心
- 实验时间：每周四13:00-15:15,3课时,8次实验课
- 闭卷考试2学时　　时间、地点待定

课程内容

- 回顾计算结构（上）接口电路的内容，设计并仿真验证一个POC电路。
- 回顾计算结构（上）微程序控制的内容，设计仿真并硬件下载验证一个简单CPU系统的设计，要求其中控制器使用微程序方式完成。

I/O模块

- 在计算机中，I/O模块承担在不同设备之间传递控制和数据信号的任务。针对CPU而言，I/O模块承担着CPU和外设的接口工作。

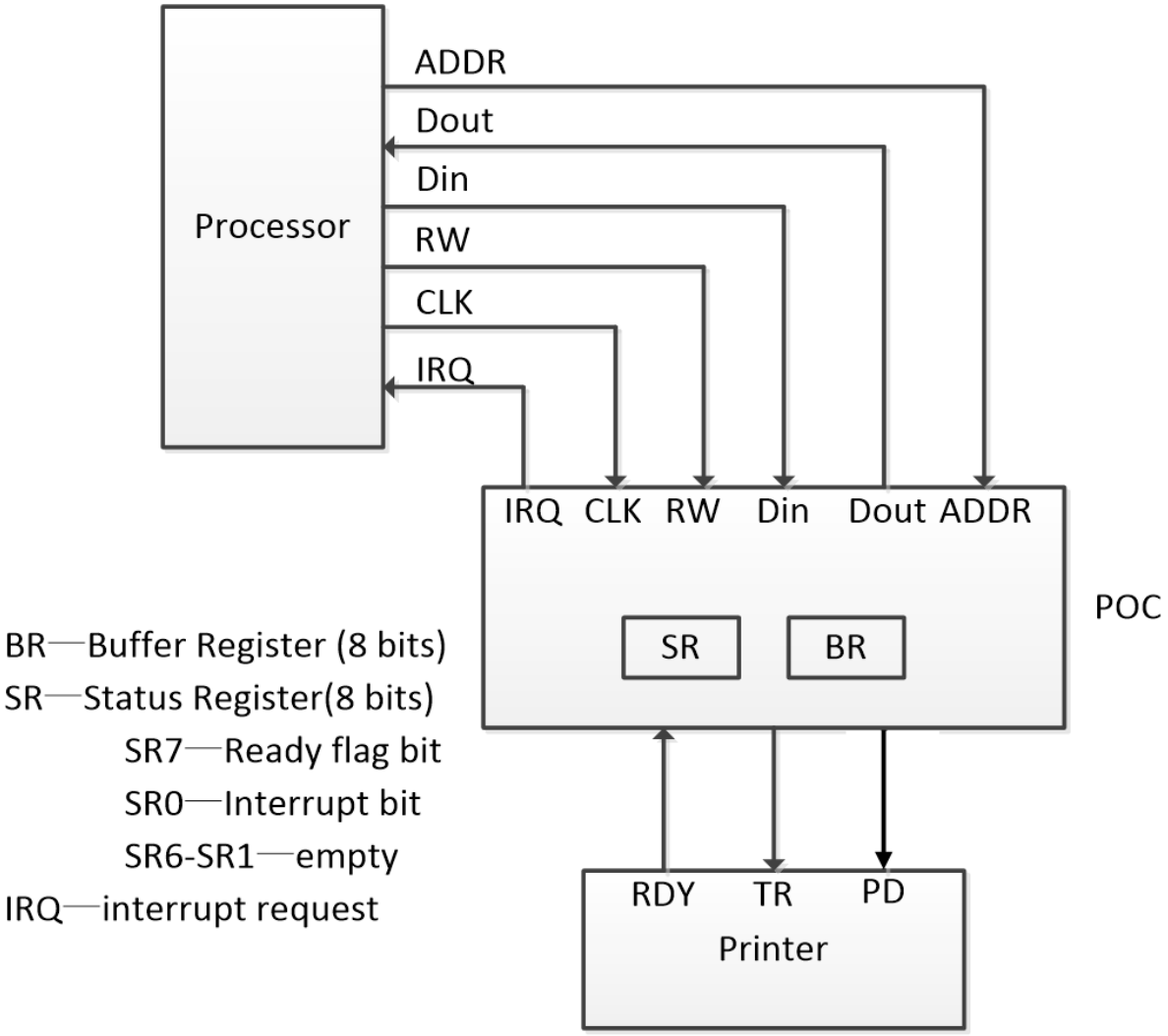
  I/O模块承担的工作经常包括：

  - 指令译码
  - 数据
  - 状态报告
  - 地址识别

I/O操作

- I/O操作的三种技术方式
  - 编程I/O　CPU对I/O发出命令，I/O模块接收命令并相应动作。CPU需要周期检查I/O模块状态。效率较低。
  - 中断I/O　I/O模块准备好时，会发送中断信号给CPU。效率较高。
  - DMA　I/O模块不经过CPU直接从内存存取数据，减轻CPU的资源占用率。

POC设计



BR—Buffer Register (8 bits)
SR—Status Register(8 bits)
SR7—Ready flag bit
SR0—Interrupt bit
SR6-SR1—empty
IRQ—interrupt request

POC设计

- CPU与POC接口
  - 数据，地址，读写控制，时钟，中断请求
  - 查询方式：SR0一直为0.

CPU通过合适的地址选中SR寄存器，查询SR7信息，如果SR7=1，CPU选中BR寄存器，将要打印的一个字节的数据写入BR,完成后CPU将SR7寄存器置为0，表明CPU已经写入新数据且尚未被处理。POC如果检测到SR7寄存器被置为0，开始与外设（打印机）握手操作，操作完成后POC将SR7寄存器置为1，即“准备好”状态。

POC设计

- CPU与POC接口
  - 中断方式：SR0一直为1.

POC将数据送至打印机后，除将SR7置为1（准备好），表明发送中断请求IRQ信号，CPU收到IRQ信号后，不再查询SR7，直接选中BR，将数据写入BR，然后CPU将SR7置为0，表明CPU已经写入新数据且尚未被处理。POC如果检测到SR7被置为0，表明收到新数据，开始与外设（打印机）握手操作，操作完成后POC将SR7置为1，由于SR0=1，使得IRQ信号拉低为低电平0，即发出中断请求。

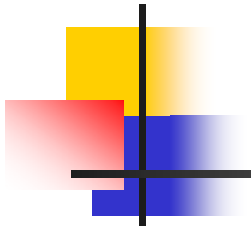
POC设计

- POC与打印机接口
  - 当打印机准备好接收新的数据时，打印机将RDY置为1，等待新的数据从POC送来。POC完成与CPU的握手后，将数据送到PD端口。POC检测到打印机的RDY=1，在TR发送脉冲，表明发送请求，打印机检测到TR后，将RDY置为0，接收PD的数据送至打印。延迟一段时间，打印完成后，打印机又将RDY置为1，表明准备好。

POC设计要求

- POC模块必须同时支持查询方式和中断方式，可以根据需要切换选择两种方式中的一种。
- 为了有效的仿真验证，建议设计一个Processor模块，与POC联合进行仿真。
- 打印机需要单独设计，用来配合POC的验证。
- 两人一组，合作完成。每人需掌握全部设计内容。
- 完成后撰写实验报告，每组1份。提交信箱：coareport@163.com
- 报告于第5周上课前提交。提交格式：Word或PDF格式。
- 提交邮件请按下列主题标注：

  计算结构POC报告***姓名。（注：请将***用自己的学号替代，将“姓名”用自己的姓名替代）。



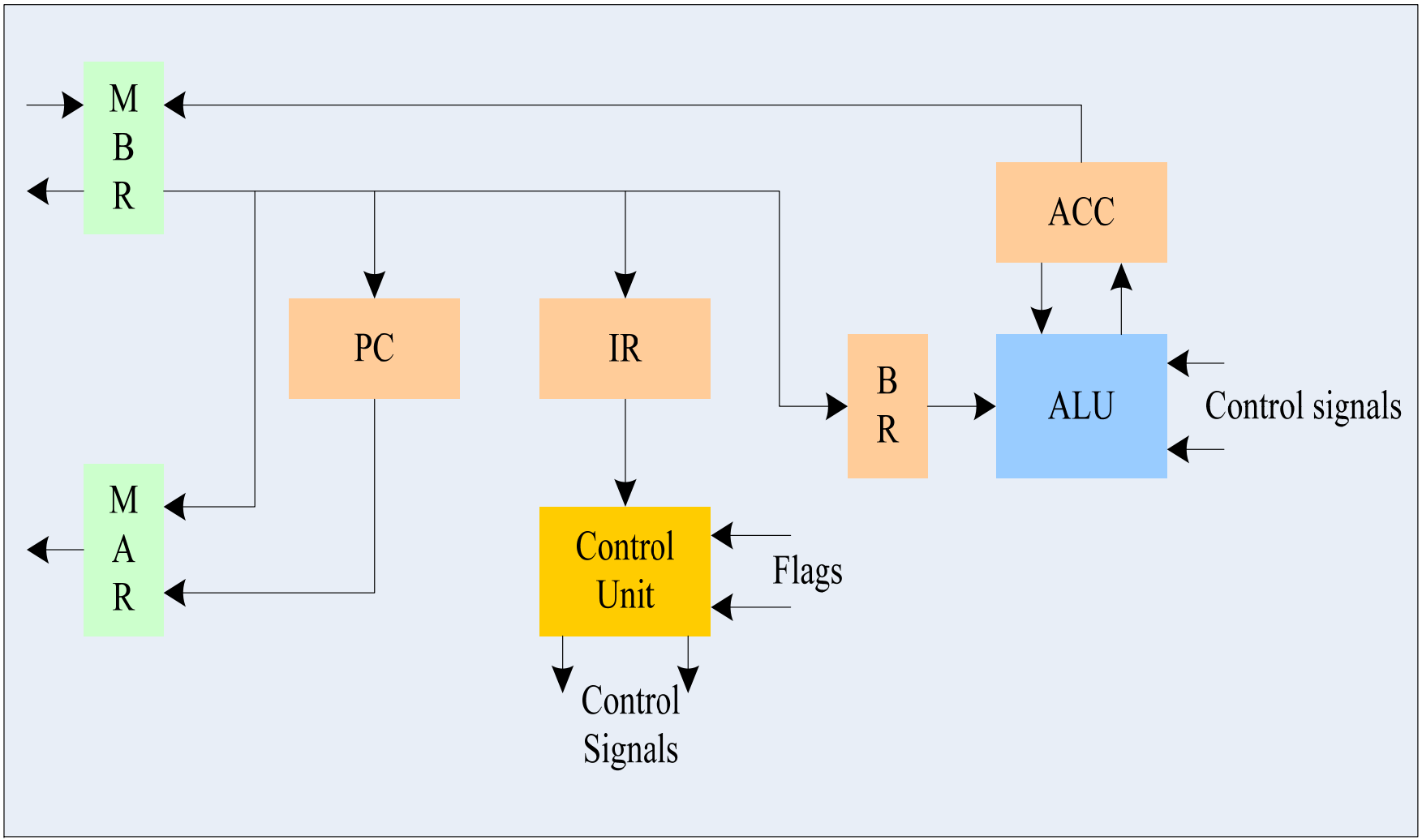
CPU设计

- 此部分要求设计一个简单的CPU。该CPU拥有基本的指令集，并且能够使用指令集运行简单的程序。另外，CPU的控制器部分（CU）要求必须采用微程序设计方式。

CPU结构

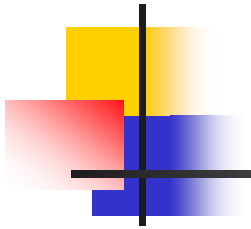
- 取指：CPU要从存储器中读取指令。
- 译码：必须翻译指令用以确定要执行的操作。
- 取数据：指令的执行可能会要求从存储器或I/O模块中读取数据。
- 处理数据：指令的执行可能会要求对数据进行算术或逻辑运算操作
- 写数据：指令执行的结果可能需要写入存储器或者I/O模块中。

CPU内部结构


M
B
R
ACC
PC
IR
B
R
ALU
Control signals
M
A
R
Control
Unit
Flags
Control
Signals

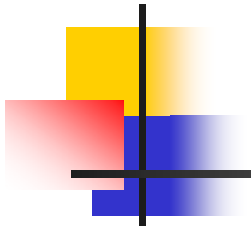
CPU内部寄存器

- MAR（Memory Address Register）
  - MAR存放着要从存储器中读取或要写入存储器的存储器地址。
  - 此处，“读”定义为CPU从内存中读。“写”定义为CPU把数据写入内存。
  - 本课程的设计中，MAR拥有8比特，可以存取256个地址。



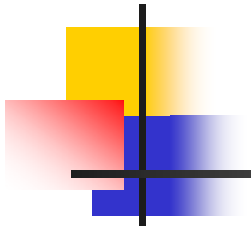
CPU内部寄存器

- MBR（Memory Buffer Register）
  - MBR存储着将要被存入内存或者最后一次从内存中读出来的数值。
  - 本课程的设计中，MBR有16比特。
- PC（Program Counter）
  - PC寄存器用来跟踪程序中将要使用的指令。
  - 本课程中，PC有8比特。



CPU内部寄存器

- IR（Instruction Register）
  - IR存放指令的OPCODE（操作码）部分。
  - 本课程中，IR有8比特。
- BR（Buffer Register）
  - BR作为ALU的一个输入，存放着ALU的一个操作数。本课程中，BR有16比特。
- ACC（Accumulator）
  - ACC保存着ALU的另一个操作数，而且通常ACC存放着ALU的计算结果。本课程中，ACC有16比特。

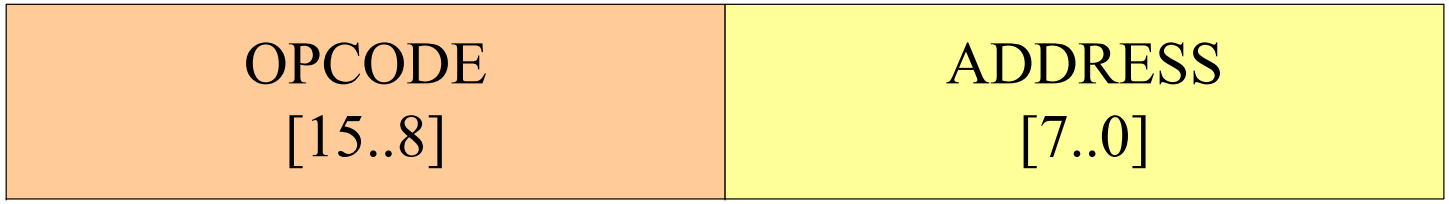


CPU指令集

- 本课程中，采用单地址的指令集结构。指令字包括两部分：操作码（OPCODE），用来定义指令的功能；地址段(Address Part)，用来存放要被操作的指令的地址。称之为直接寻址（Direct Addressing）。在一些少量的指令中，地址段就是操作数，这是立即数寻址（Immediate Addressing）。

CPU指令集

- 简化起见，内存的大小为256*16. 指令字有16比特，其中操作码部分8比特，地址段8比特。



CPU指令集

Table 1 List of instructions and relevant opcodes

INSTRUCTION	OPCODE	COMMENTS
STORE X	00000001	ACC → [X]
LOAD X	00000010	[X] → ACC
ADD X	00000011	ACC + [X] → ACC
SUB X	00000100	ACC − [X] → ACC
JMPGEZ X	00000101	If ACC ≥ 0 then X → PC else PC+1 → PC
JMP X	00000110	X → PC
HALT	00000111	Halt a program
MPY X	00001000	ACC × [X] → ACC, MR
DIV X	00001001	ACC ÷ [X] → ACC, DR
AND X	00001010	ACC and [X] → ACC
OR X	00001011	ACC or [X] → ACC
NOT X	00001100	NOT [X] → ACC
SHIFTR	00001101	SHIFT ACC to Right 1bit, Logic Shift
SHIFTL	00001110	SHIFT ACC to Left 1bit, Logic Shift
……	……	……

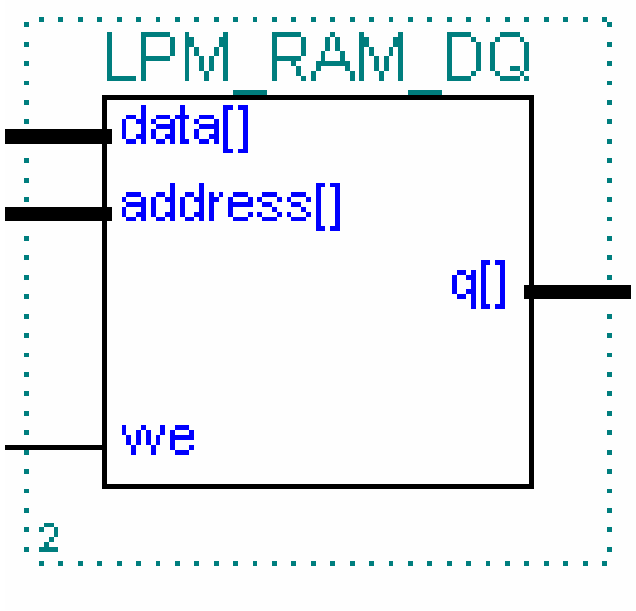
ALU

- ALU是用来执行算术和逻辑操作的单元。几乎所有的操作都是将相应的数据带到ALU来进行处理，然后把结果取出。

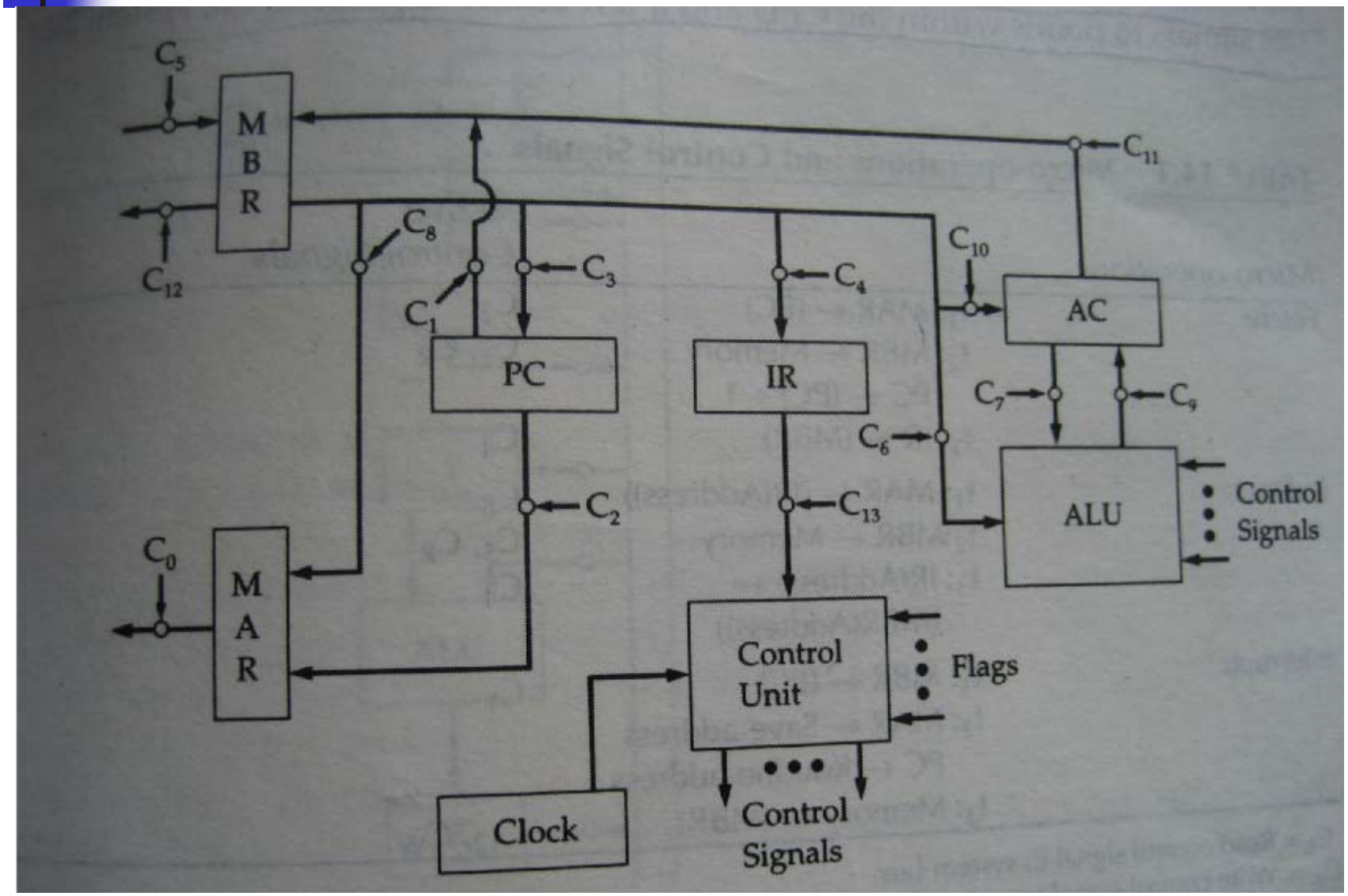
Table 3 ALU Operations

Operations	Explanations
ADD	(ACC)⇐(ACC)+(BR)
SUB	(ACC)⇐(ACC)–(BR)
AND	(ACC)⇐(ACC) and (BR)
OR	(ACC)⇐(ACC) or (BR)
NOT	(ACC)⇐Not (BR)
SRL	(ACC)⇐Shift (ACC) to Left 1 bit
SRR	(ACC)⇐Shift (ACC) to Right 1 bit

存储器（Memory）



微程序控制器设计


C5
MBR
C11
C12
C8
C1
C3
C4
C10
AC
PC
IR
C7
C9
C6
ALU
Control
Signals
C2
C13
C0
MAR
Control
Unit
Flags
Clock
Control
Signals

控制器内部结构

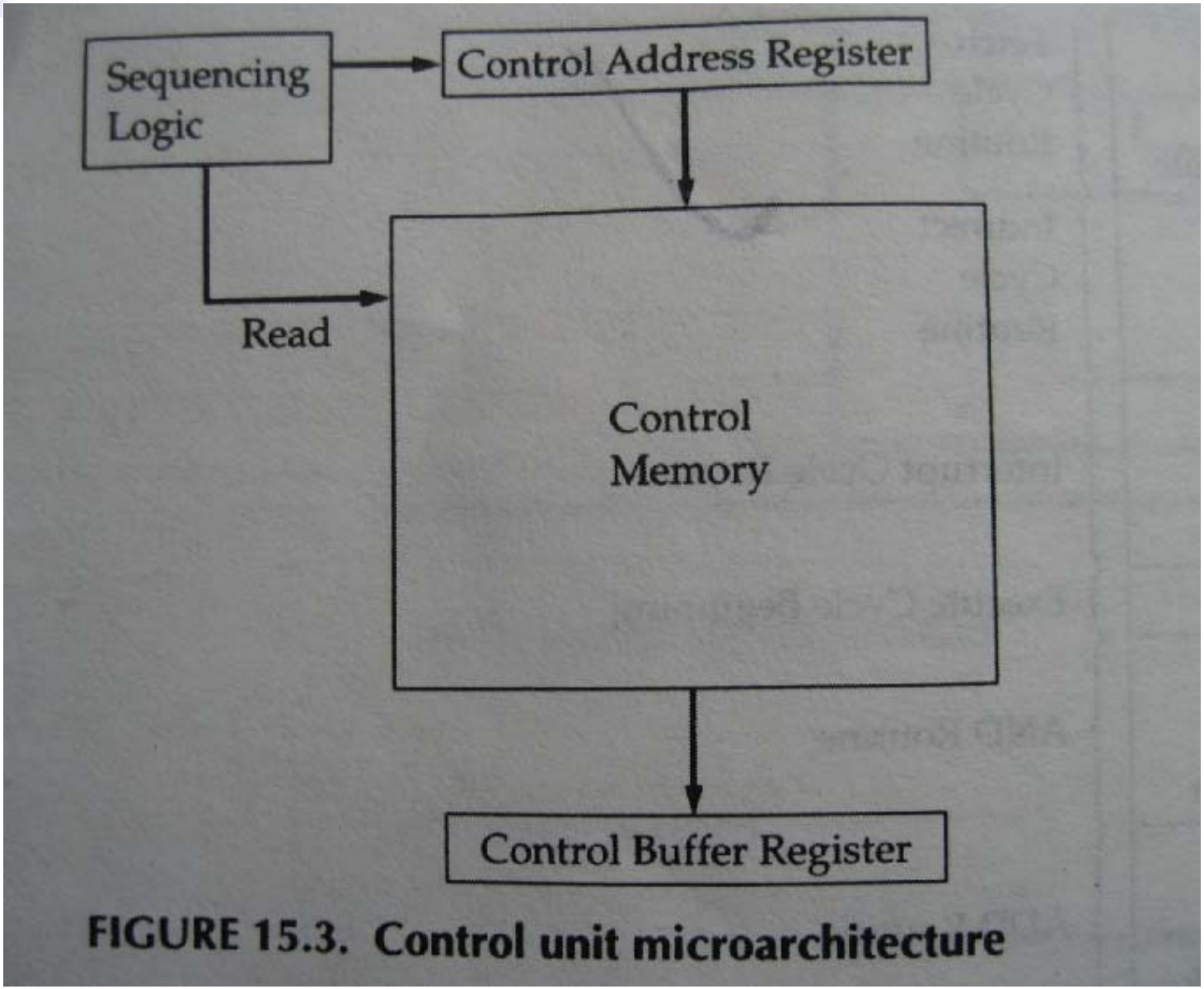


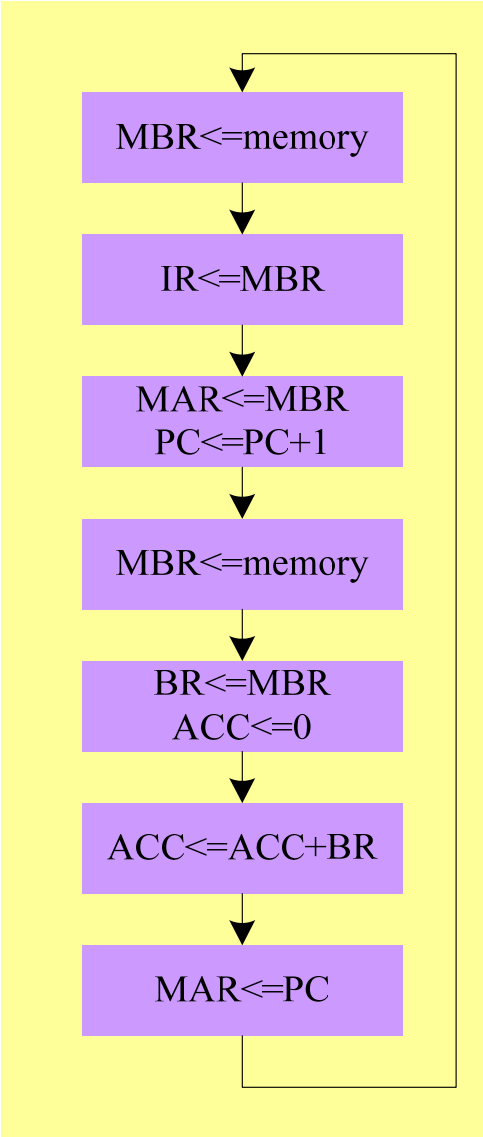
FIGURE 15.3. Control unit microarchitecture

微程序控制器设计

- 控制器的控制存储器（CM）中存放有每一个指令对应的微程序，微程序包含若干行，每行都是一个微指令。0和1代表着断和通。对每一个微指令而言，控制器做的就是生成一系列控制信号来控制相关寄存器的操作。
- 控制地址寄存器（CAR）控制着下面要读取哪一条微指令，也就是读取哪一个地址，从CM中读取了一条微指令就相当于执行了若干个控制信号。

控制器设计

- 需要根据CPU的结构和具体设计来决定实际需要的控制信号，下面给出一个例子用来体现该过程。该例是LOAD指令的设计。



控制器设计

Table 4 Some Control signals for the LOAD instruction

Bit in Control Memory	*Micro-operation*	*Meaning*
C0	CAR<=CAR+1	Control Address Increment
C1	CAR<=***	Control Address Redirection, depends on the position of microinstruction
C2	CAR<=0	Reset Control Address to zero position
C3	MBR<=memory	Memory Content to MBR
C4	IR<=MBR[15..8]	Copy MBR[15..8] to IR for OPCODE
C5	MAR<=MBR[7..0]	Copy MBR[7..0] to MAR for address
C6	PC<=PC+1	Increment PC for indicating position
C7	BR<=MBR	Copy MBR data to BR for buffer to ALU
C8	ACC<=0	Reset ACC register to zero
C9	ACC<=ACC+BR	Add BR to ACC
C10	MAR<=PC	Copy PC value to MAR for next address
…	……	……

控制器设计

Table 5 Microprogram for LOAD instruction

Microprogram	*Control signals*
MBR<=memory, CAR<=CAR+1	C3, C0
IR<=MBR[15..8], CAR<=CAR+1	C4, C0
CAR<=*** (*** is determined by OPCODE)	C1
MAR<=MBR[7..0], PC<=PC+1, CAR<=CAR+1	C5, C6, C0
MBR<=memory, CAR<=CAR+1	C3, C0
BR<=MBR, ACC<=0, CAR<=CAR+1	C7, C8, C0
ACC<=ACC+BR, CAR<=CAR+1	C9, C0
MAR<=PC, CAR<=0	C10,C2

CPU设计要求

- 独立设计微程序控制器及外围的各寄存器。
- 使用实验指导书中的1+2+…+100和相应的乘法例子来验证程序的正确性与完整性。
- 要求完成并支持指令集中列出的除了除法外的所有指令。
- 不得随意增加CPU内寄存器，不能随意增加控制器到各寄存器的控制线。
- 必须采用微程序方式设计控制器，否则不予通过。

CPU设计要求

- CPU设计要求仿真和硬件下载都需要验证。
- 下载硬件采用Xilinx 开发板。可以设计使用其中的按键、开关、LED灯、数码管等配合控制与显示。

CPU设计要求

- 完成后撰写实验报告，每组1份。提交信箱：coareport@163.com
- 2-3人一组，合作完成。每人需掌握全部设计内容。
- 报告于课程结束后1周内提交。提交格式：Word或PDF格式。
- 提交邮件请按下列主题标注：

  计算结构CPU报告***姓名。（注：请将***用自己的学号替代，将“姓名”用自己的姓名替代）。