处理器设计

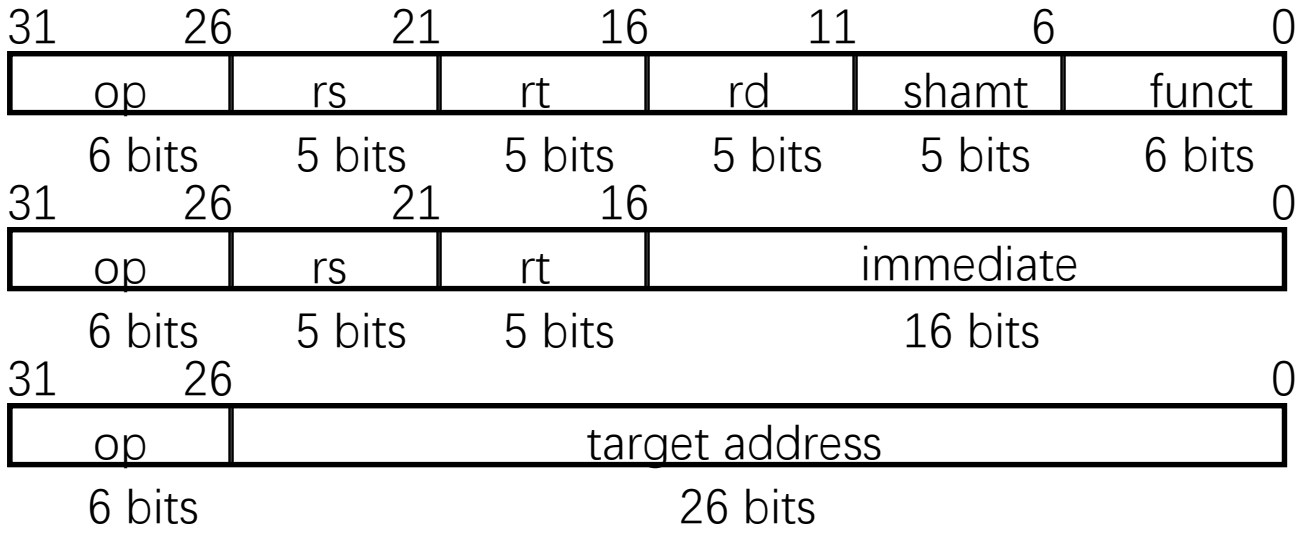
1. 处理器设计的步骤

主讲人: 邓倩妮
上海交通大学

大部分内容来自于：
Computer Organization and Design, 4th Edition, Patterson & Hennessy,

回顾：MIPS 指令格式

- 32 位，三种格式:
  - R-type
  - I-type
  - J-type



- 各部分含义:
  - op: 指令操作码
  - rs, rt, rd: 源寄存器、目标寄存器标识
  - shamt: shift amount 移位量
  - funct: 对相同op的指令进一步功能细化
  - address / immediate: 地址偏移量（address offset）or 立即数（immediate value）
  - target address: jump 指令的跳转目标地址

处理器设计

- 简化的 MIPS 指令实现

  访问存储器: **lw, sw**

  算术逻辑运算指令: **add, sub, or**

  控制流指令: **beq, j**

各条指令实现步骤和完成的功能

- 一开始，都要先取指令：

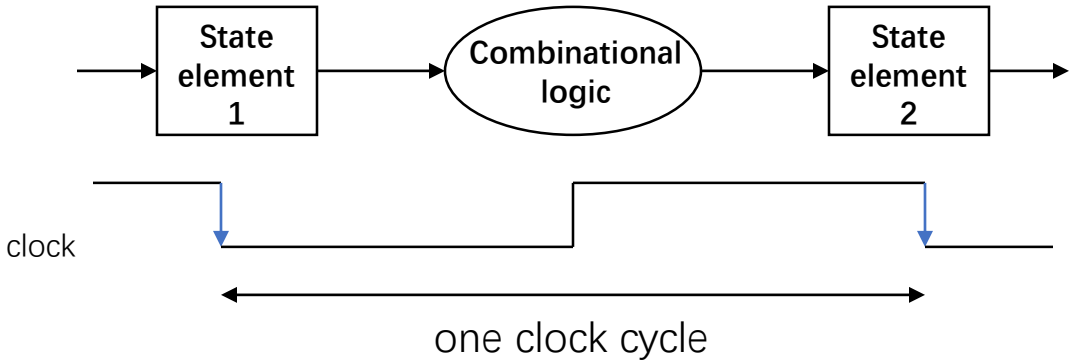
op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADDU	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUBU	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORi	R[rt] <– R[rs] | zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then	PC <– PC + 4 +sign_ext(Imm16)] || 00
	else PC <– PC + 4	

Clocking Methodologies

- 典型实现：单周期处理器，一个周期完成一条指令：
  - 读状态单元的内容 -> 通过组合逻辑电路实现指令的功能 -> 将结果写入一个或多个状态单元



State elements 状态单元 ：

存储单元，例如：register（寄存器）

Edge-triggered 边沿触发

状态转换发生在时钟边沿 (clock edge)

- 假设：状态单元每一周期更新一次，是否更新，需要一个显式的控制信号
- 仅仅在时钟边沿来到，同时写允许信号有效时(write control is asserted)才更新状态单元

设计处理器的五个步骤

1. 分析指令系统， 得出对数据通路的需求
   - The meaning of each instruction is given by the register transfers
   - 例如：ADDU指令的数据通路需求：R[rd] <– R[rs] + R[rt];
2. 选择数据通路上合适的组件
   - 例如：加法器、 算术逻辑运算单元、寄存器堆、控制器
3. 连接组件构成数据通路
4. 分析每一条指令的实现，以确定控制信号，
   - 不同的控制信号影响寄存器之间的数据传送
5. 集成控制信号，完成控制逻辑

Step1: 分析各条指令

- 一开始，都要先取指令：

op | rs | rt | rd | shamt | funct = MEM[PC]

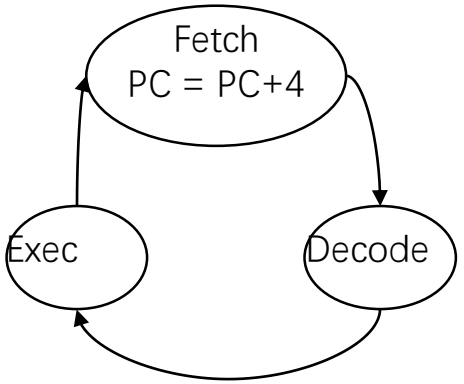
op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADDU	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUBU	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORi	R[rt] <– R[rs] | zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4

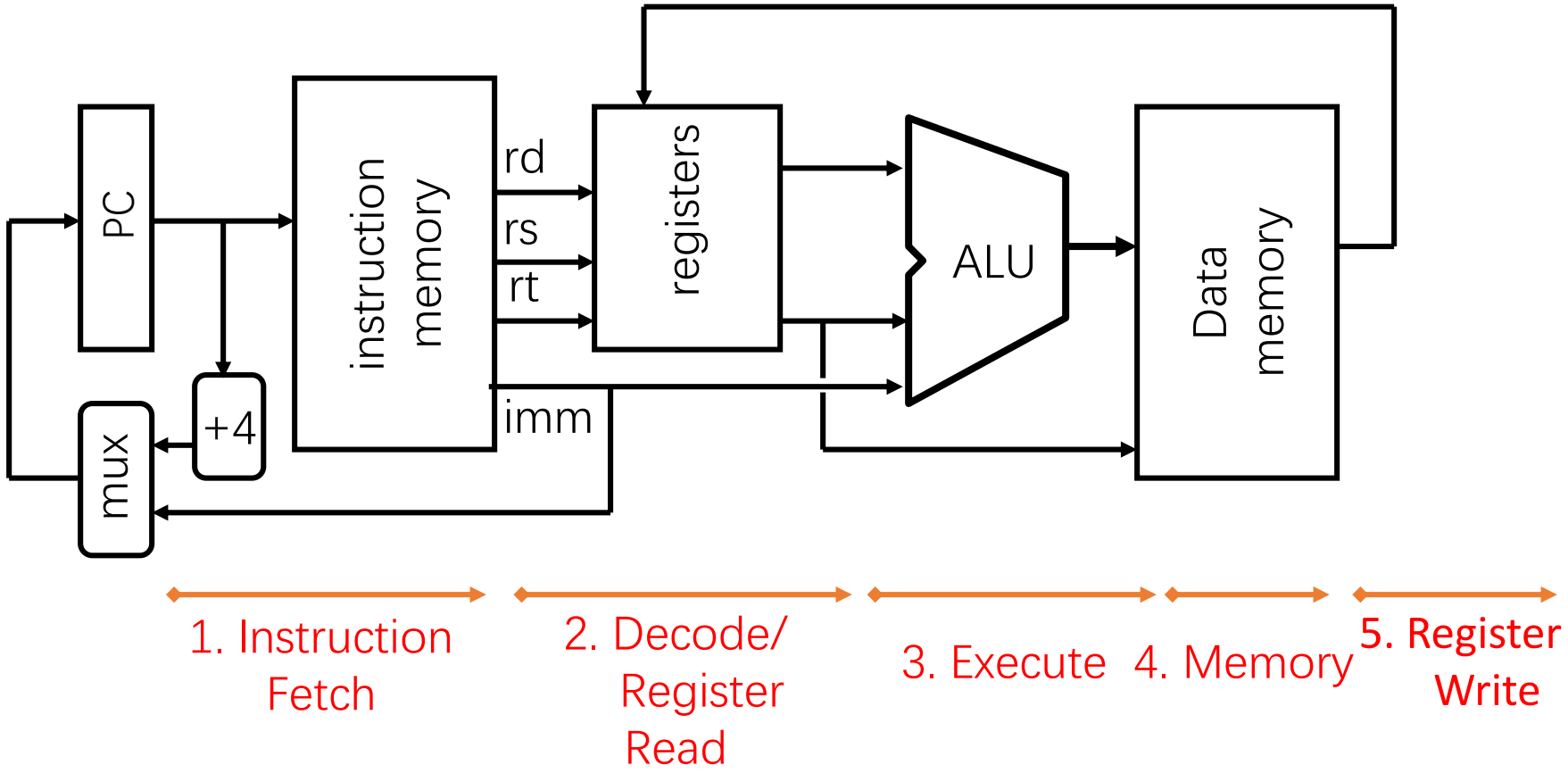
BEQ if (R[rs] == R[rt]) then PC <– PC + 4 +sign_ext(Imm16)] || 00
else PC <– PC + 4

指令的执行过程

取指令
指令译码
指令执行



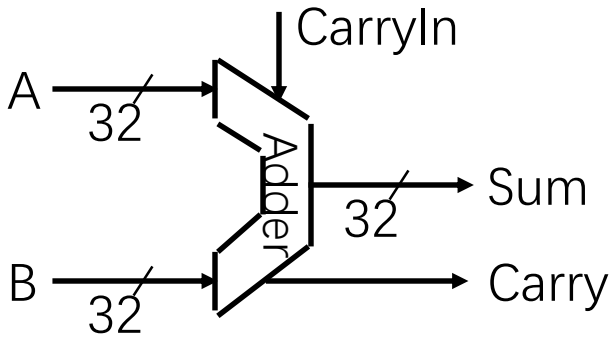
数据通路 Datapath 的大致需求

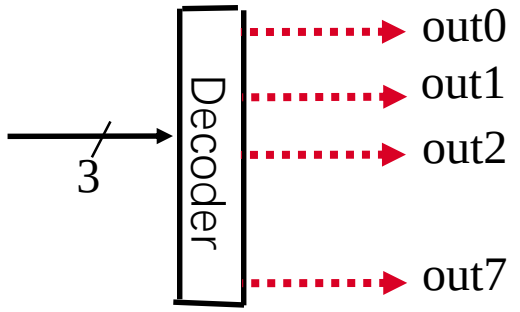


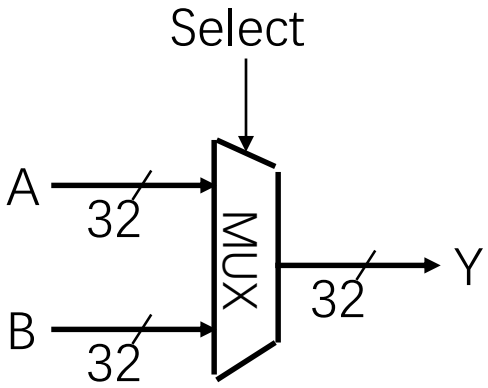
Step 2: 选择数据通路上合适的组件

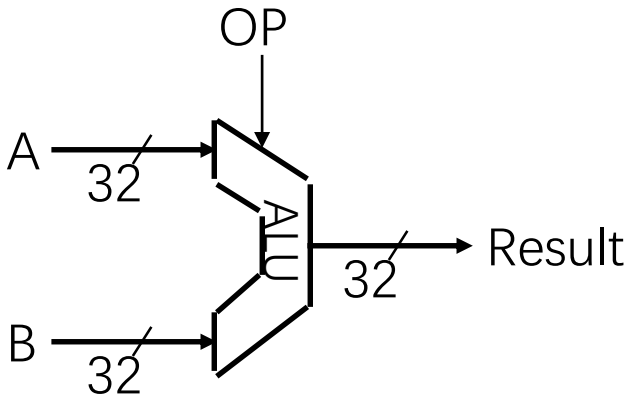
- 组合单元 Combinational Elements
  - 实现指令的逻辑功能
- 状态单元 Storage Elements
  - 时钟边沿触发状态更新 Clocking methodology

组合单元 Combinational Logic Elements (基本模块)



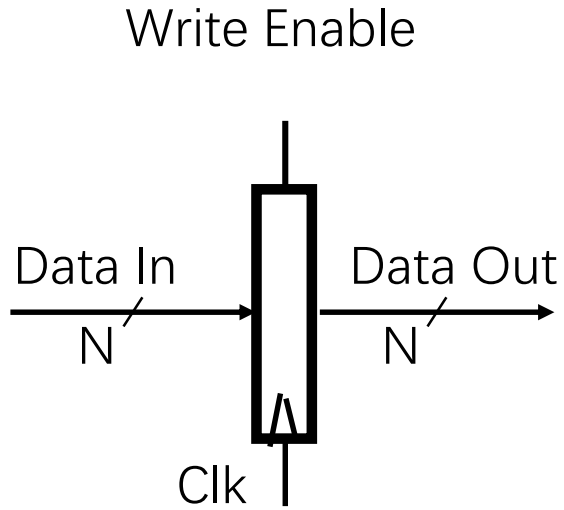






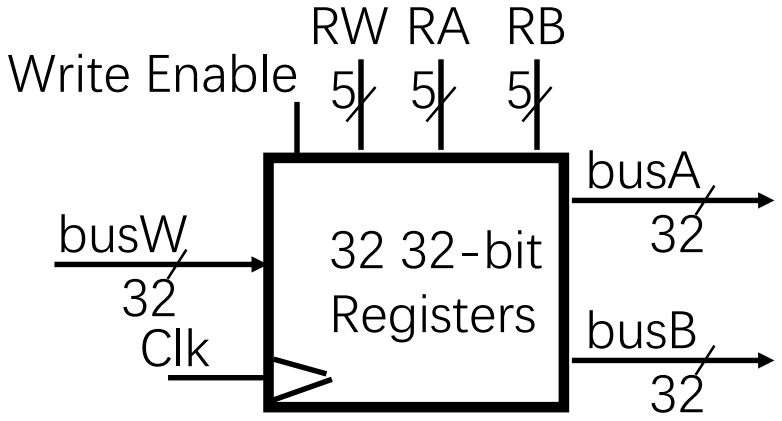
状态单元 :寄存器 Register (基本模块)

- 寄存器 Register
  - N-bit input and output
  - Write Enable input
  - Write Enable:
    - negated (0): Data Out will not change
    - asserted (1): Data Out will become Data In



Storage Element: Register File 寄存器文件

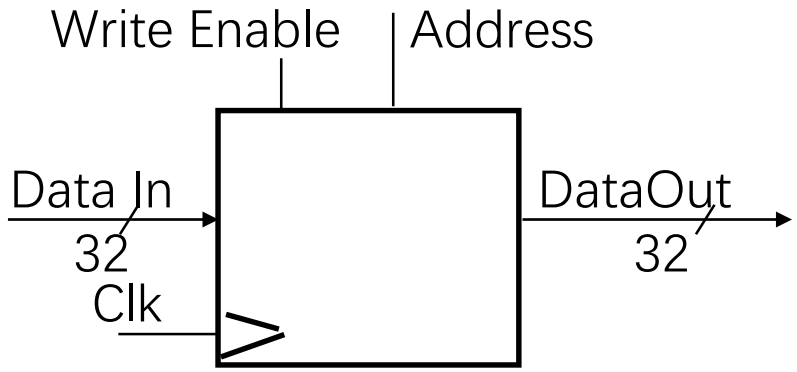
- Register File consists of 32 registers: 32个寄存器
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW
- Register is selected by: 寄存器选通
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1



- Clock input (CLK) 时钟输入
  - The CLK input is a factor ONLY during write operation （只在写寄存器时需要时钟边沿触发）
  - During read operation, behaves as a combinational logic block: （读操作，可以看做一个组合逻辑电路模块的实现）
    - RA or RB valid => busA or busB valid after “access time.”

存储器

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by: 存储字选择
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus



- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation （只在写存储器时需要时钟边沿触发）
  - During read operation, behaves as a combinational logic block: （读操作，可以看做一个组合逻辑电路模块的实现）
    - Address valid => Data Out valid after “access time.”
    - 一定访问时间内完成从“地址信号有效” =>“数据输出”

小结：设计处理器的五个步骤

1. 分析指令系统， 得出对数据通路的需求
   - The meaning of each instruction is given by the register transfers
   - 例如：ADDU指令的数据通路需求：R[rd] <– R[rs] + R[rt];
2. 选择数据通路上合适的组件
   - 例如：加法器？ 算术逻辑运算单元？寄存器堆？存储器？
3. 连接组件构成数据通路
4. 分析每一条指令的实现，以确定控制信号，
   - 不同的控制信号影响寄存器之间的数据传送
5. 集成控制信号，完成控制逻辑

谢谢！