Introduction

CPU Time = Instruction Count × CPI × Clock Cycle Time

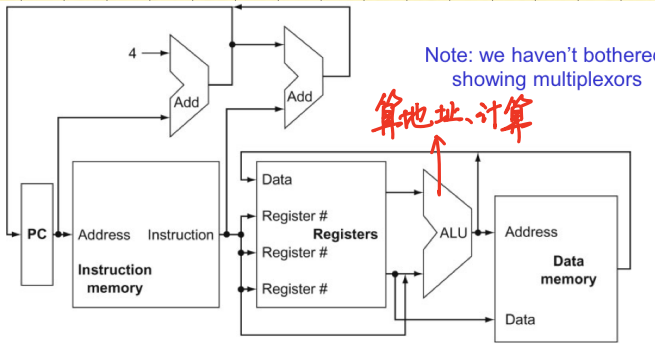
- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two MIPS implementations
  - A simplified version → 在一个周期
  - A more realistic pipelined version → 在多个周期

Basic MIPS Architecture

- We have known the instructions a CPU should execute, we'll design a simple CPU that executes:
  - basic math (add, sub, and, or, slt)
  - memory access (lw and sw)
  - branch and jump instructions (beq and j)

Implement overview

- We need memory
  - to store instructions
  - to store data
  - for now, let's make them separate units
- We need registers, ALU, and a whole lot of control logic
- CPU operations common to all instructions:
  - use the program counter (PC) to pull instruction out of instruction memory
  - read register values



Source: H&P textbook

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

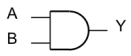
Instruction memory 和 data memory 实际上是CPU中的cache，便于快速取指和数.

Logic Design Basic

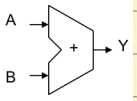
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

Combinational Elements

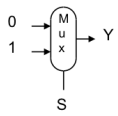
- And gate
  - Y = A & B



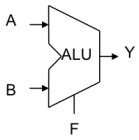
- Adder
  - Y = A + B



- Multiplexer
  - Y = S ? 1 : 0

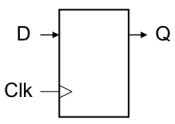


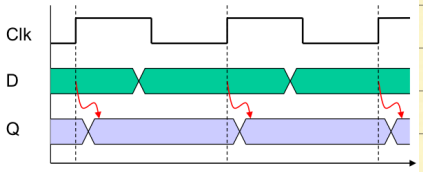
- Arithmetic/Logic Unit
  - Y = F(A, B)



State (sequential) Elements

- State element
  - The state element has a pre-stored state
  - It has some internal storage
  - Has at least two inputs and one output (e.g. D-type flip-flop):
    - The data to be written into the element
    - The clock which determines when the data is written
    - The output: the value that was written in earlier cycle
  - Examples: register and memory

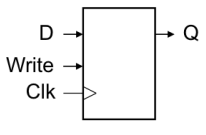


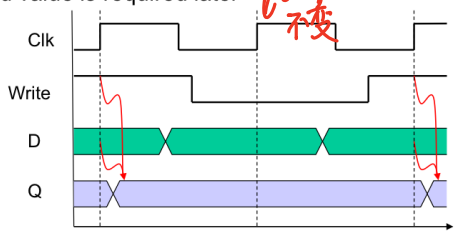


State Elements with write control

- Register without write control (e.g. program counter)
  - Uses a clock signal to determine when to update
  - Edge-triggered: update when Clk changes from 0 to 1
- Register with write control (e.g. data memory/register)
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later

Write=0 不变



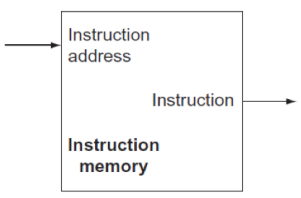


Instruction Memory

- A state element with
- Input: InstructionAddress (32-bit), clock
- Output: Instructions (32-bit)
- State: the instructions stored in the memory (n*32-bit)
- Why no control signals?
  - No write operation
  - Every clock read an instruction

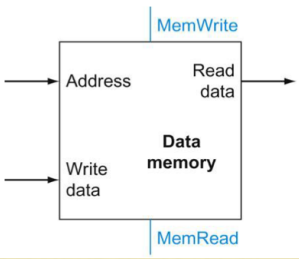
输入指令地址输出指令.

不需要使能信号，因为没有写操作.
每个时钟都读一个指令.



Data Memory

- A state element with
- Input: Address (32-bit), Write-in data (32-bit), MemWrite (1-bit), MemRead (1-bit), clock
- Output: Read-out data (32-bit)
- State: the data stored in the memory (n*32-bit)



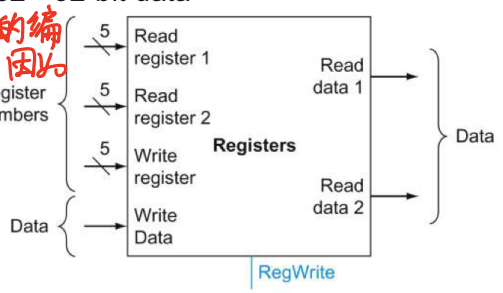
MemWrite 只当 store 指令时为1
MemRead 只当 load 指令时为1.

Registers

- State element
- Input: three register numbers (5-bit *3), write-in data (32-bit), RegWrite (1-bit)
- Output: readdata1 (32-bit), readdata 2 (32-bit)
- State: 32 * 32-bit data

32×32 bit 的数据.

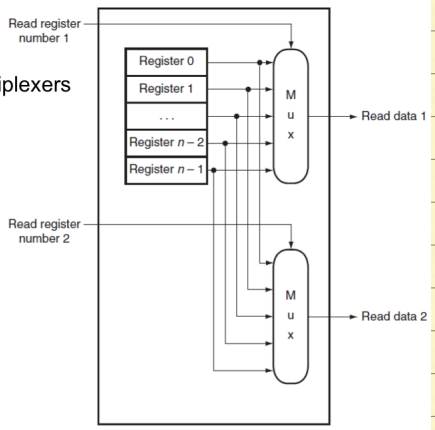
Register 的编号为5位，因为只有32个register



Register Read

- Input: two addresses
- Output: two data
- Key component: two multiplexers

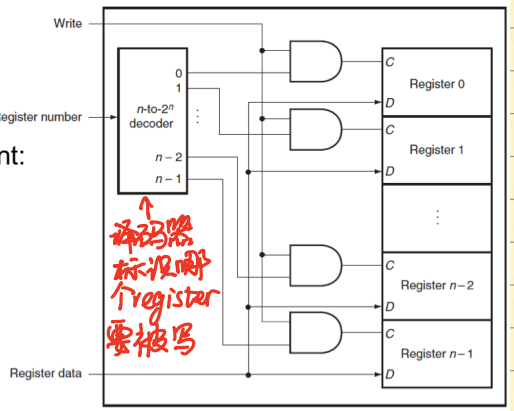
rs, rt 两个寄存器，有时需同时读.



Register Write

- Input:
  - write control
  - address
  - writing data
- Key component:
  - decoder

rd 一个寄存器

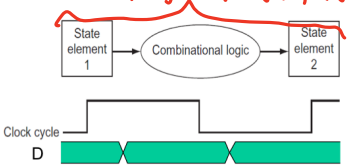


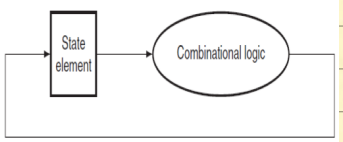
Clocking Methodology 钟控方法

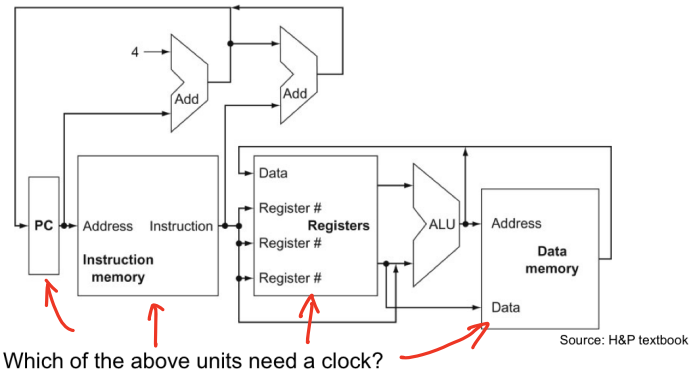
- Defines when signals can be read and when they can be written
- Edge-triggered clocking: all state changes occur on a clock edge.
- Clock time > the time needed for signals to propagate from SE1 through combinatorial element to SE2
- A state element can be read and written in the same clock cycle without creating a race, but the clock cycle should be long enough

所有状态的改变都发生在上升/下降沿.

小于一个时钟周期内稳定，避免冲突.







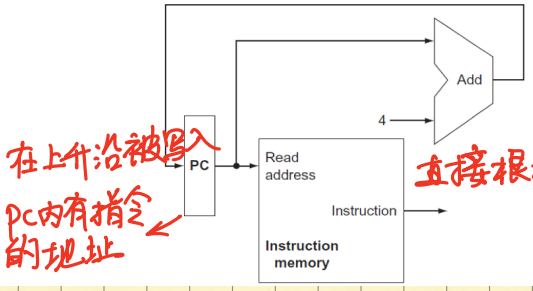
Source: H&P textbook

- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?
  Keep in mind that the latched value remains there for an entire cycle

在时钟上升沿时储存：
① PC的新值
② Register的新data

Fetch Instructions

- The instruction is fetched from I-mem and the PC is added by 4
- Components: program counter, instruction memory, adder
  - The PC is a 32-bit register, written at every positive edge of the clock, thus, it does not need write control signal.



在上升沿被写入

PC内有指令的地址

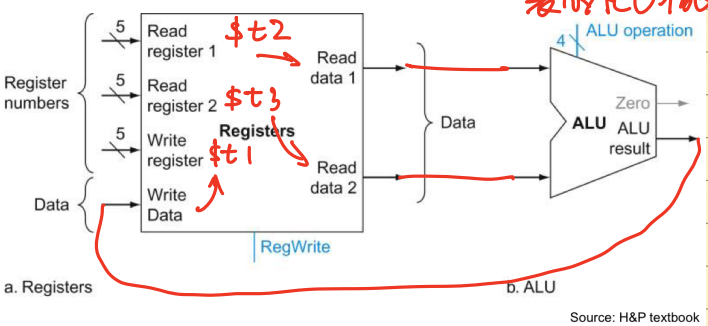
直接根据当前PC的值获取指令.

PC在每个上升沿被写，不需要写控制信号.

直到最终将data写回register时才在时钟上升沿操作.

Implementing R-type Instructions

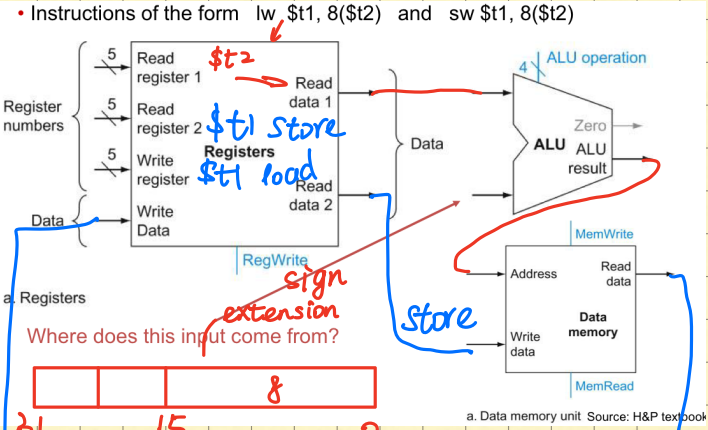
- Instructions of the form add $t1, $t2, $t3
- Explain the role of each signal



Source: H&P textbook

Implementing Loads / Stores

- Instructions of the form lw $t1, 8($t2) and sw $t1, 8($t2)



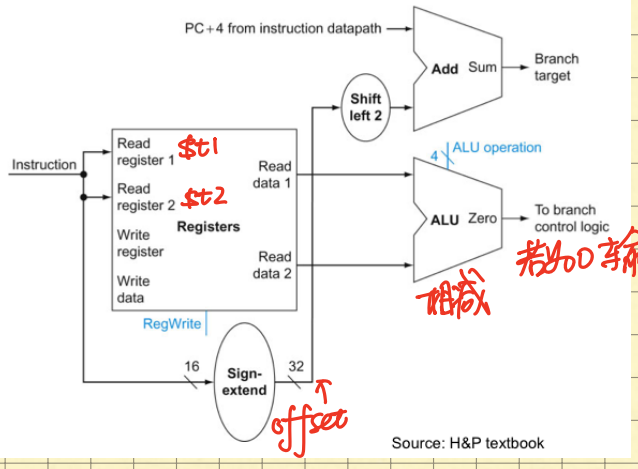
Source: H&P textbook

获取指令在上升沿

下降沿来取数据.

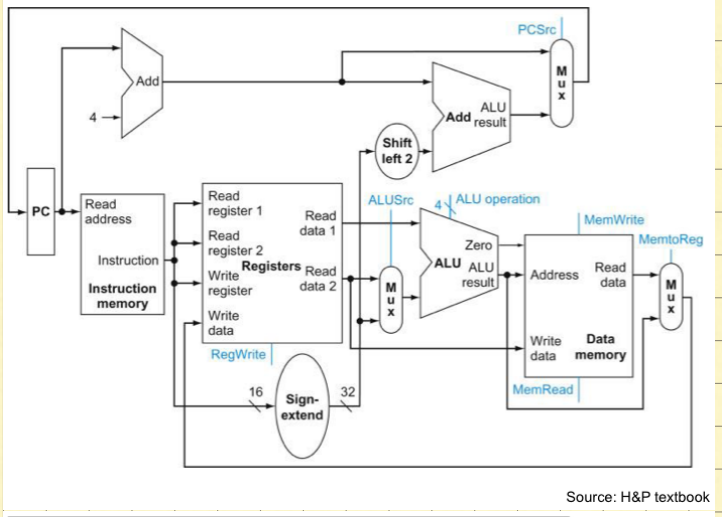
Implementing Cond-Branch Instructions

- Instructions of the form beq $t1, $t2, offset

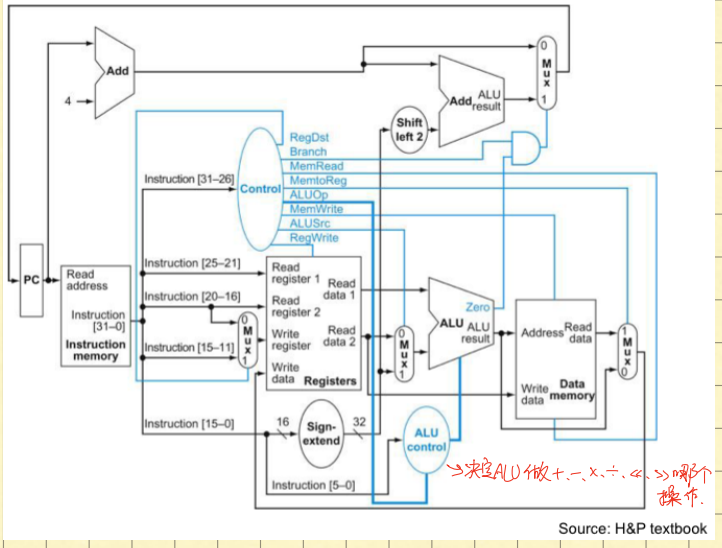


若为0等输出1，否则0.

Source: H&P textbook



Source: H&P textbook



→决定ALU做+、−、×、÷、<<、>>哪个操作.

Source: H&P textbook