MIPS-Lite Processor Datapath Design

COE608: Computer Organization and Architecture

Dr. Gul N. Khan
http://www.ee.ryerson.ca/~gnkhan
Electrical and Computer Engineering
Ryerson University

Overview

- Design a processor: step-by-step
- Requirements of the Instruction Set
- MIPS-Lite Instructions
- Components and Clocking
- Assembling an adequate Datapath
- Controlling the Datapath

Chapter 4 (4.1, 4.2 & 4.3) of the textbook

Design a Processor

Processor design (data path, alu and control)
It determines

- Clock cycle time
- Clock cycles per instruction

Single cycle processor:
Advantage:

Disadvantage:

Analyze Instruction Set
=> Data path Requirements

Meaning of each instruction is given by
register transfers

Single Cycle Processor Data path

1. Analyze the Instruction Set Interconnection to support RT

2. Select set of data path components and establish clocking methodology

3. Assemble data path meeting the requirements

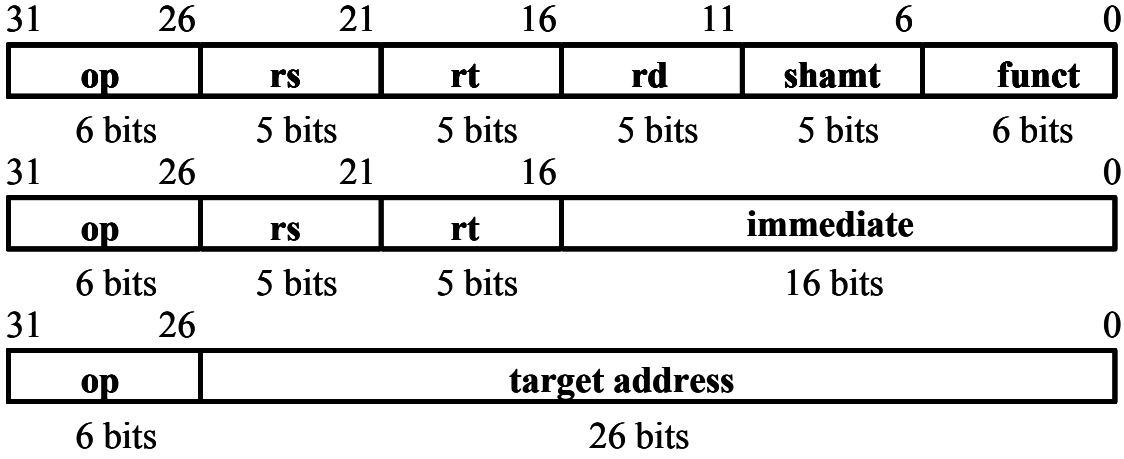
4. Analyze the implementation of each instruction.

5. Assemble the control logic.

MIPS Instruction Formats

All MIPS instructions are 32 bits long.

There are three instruction formats:



op

funct

rs, rt, rd

shamt

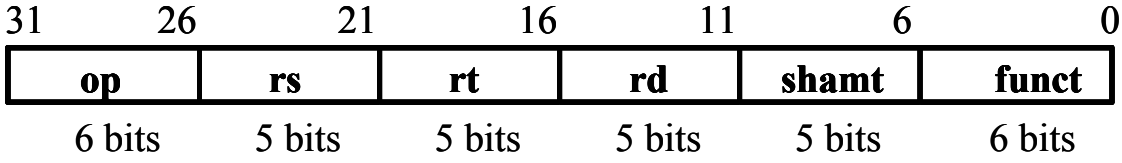
address/immediate

target address

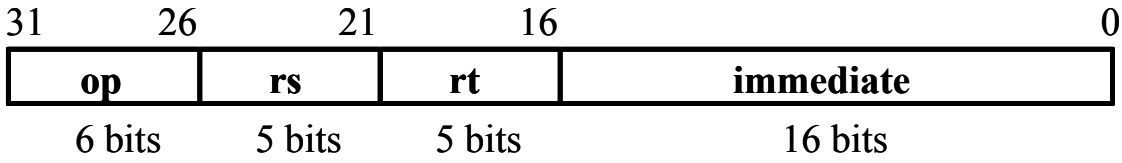
MIPS-Lite Instructions

A Subset of MIPS Instructions

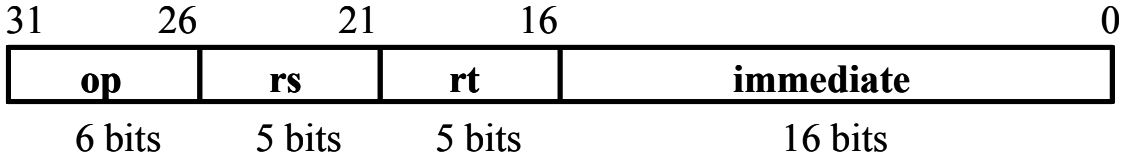
ADD and SUB



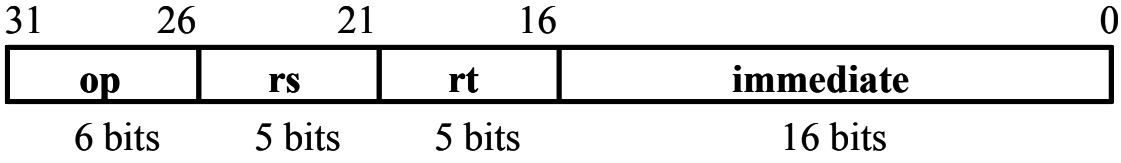
OR immediate



LOAD and STORE Word



BRANCH



Logical Register Transfers

RTL gives the meaning of the instructions

All start by fetching the instruction

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

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

inst	**Register Transfers**
ADDU	**PC <= PC + 4**
SUBU	**PC <= PC + 4**
ORi	**R[rt] <= R[rs] || zero_ext(Imm16);**
LOAD	**R[rt] <= MEM[R[rs] +sign_ext(Imm16)];**
STORE	**MEM[R[rs]+sign_ext(Imm16)] <= R[rt];**
BEQ	**if (R[rs] == R[rt]) then** **PC <= PC + 4 + {sign_ext(Imm16), 2'b 00}** **else PC <= PC + 4**

Requirements of Instruction Set

Memory

Registers (32 x 32)

PC

Extender

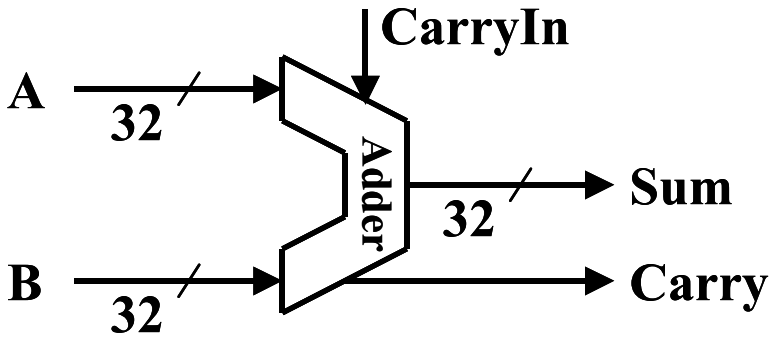
Add and Subtract register or extended immediate

Components of the Datapath

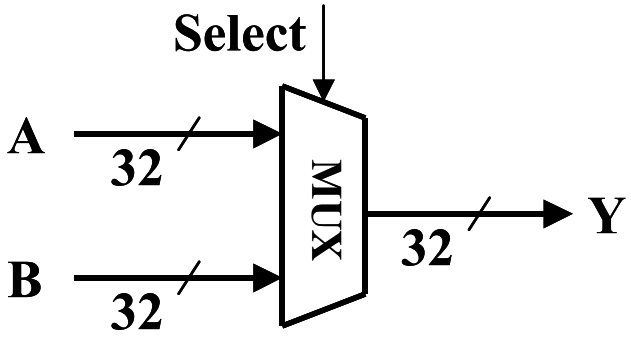
Basic Building Blocks

Combinational Logic Elements

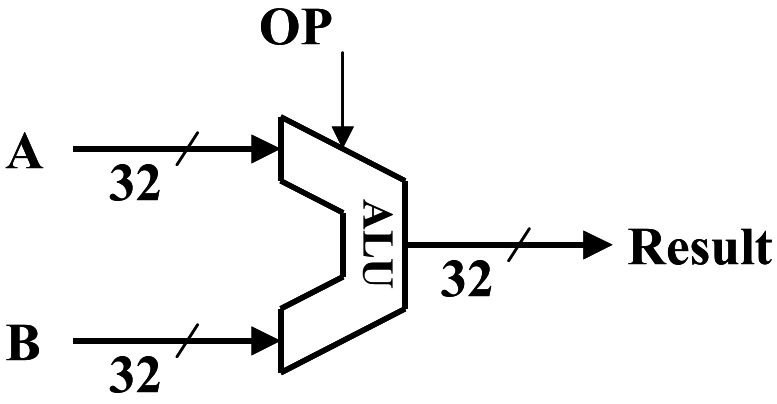
Adder



MUX



ALU

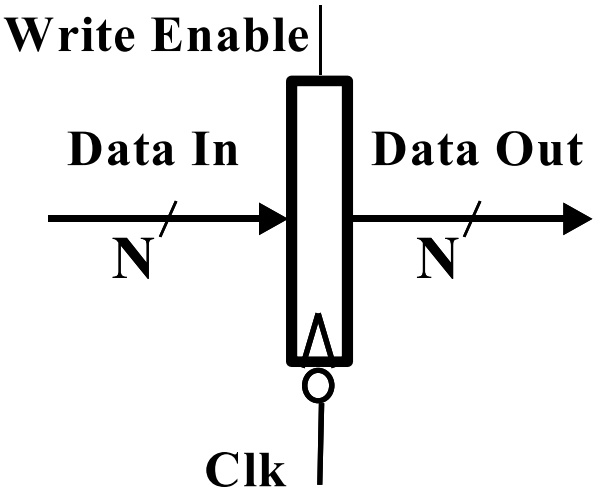


Storage Element: Register

Register

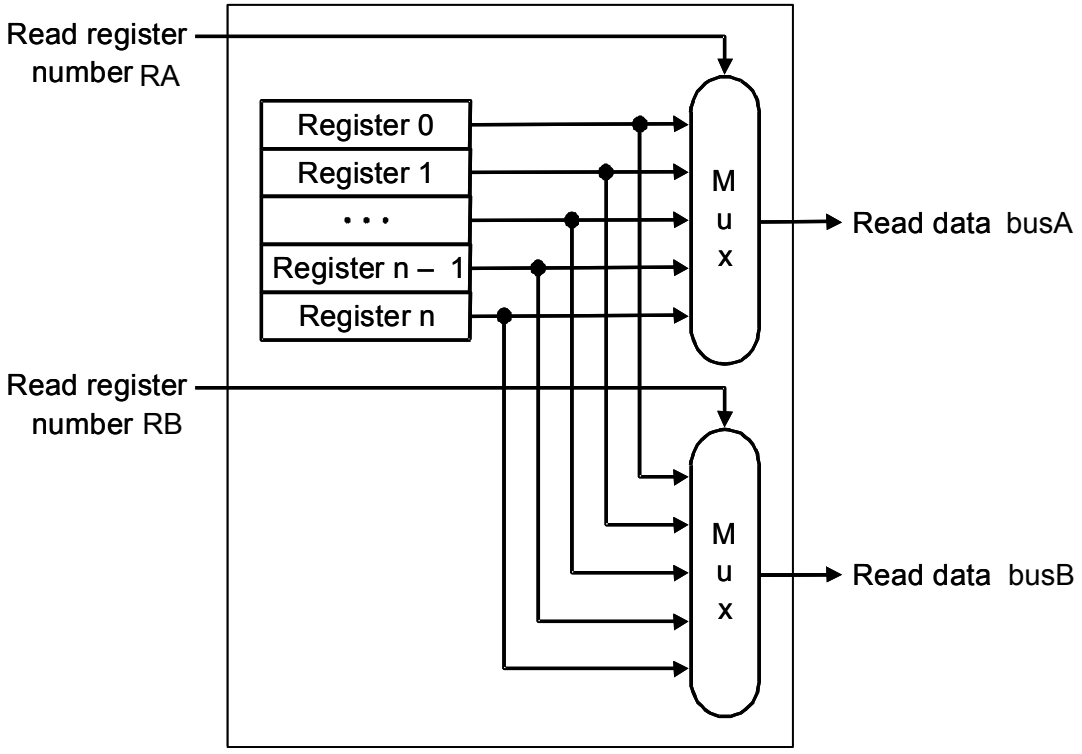
Similar to the D Flip-Flops

Write Enable:



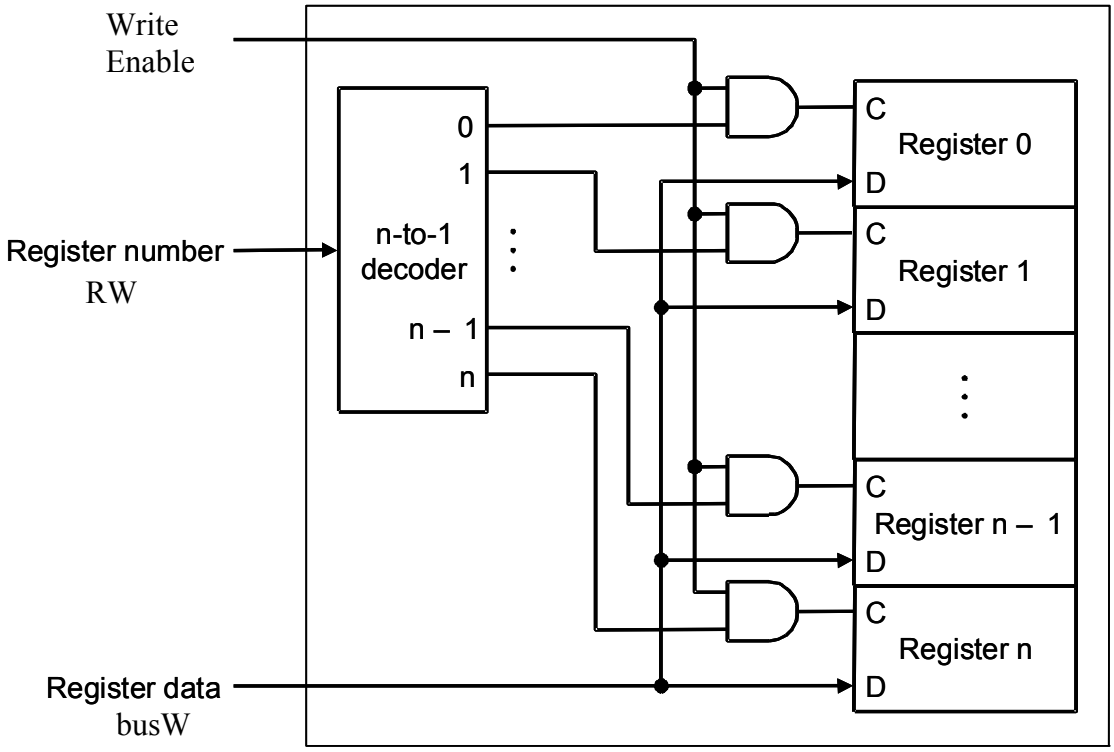
Register File

Details of Register Reading



Register File

Writing into a Register



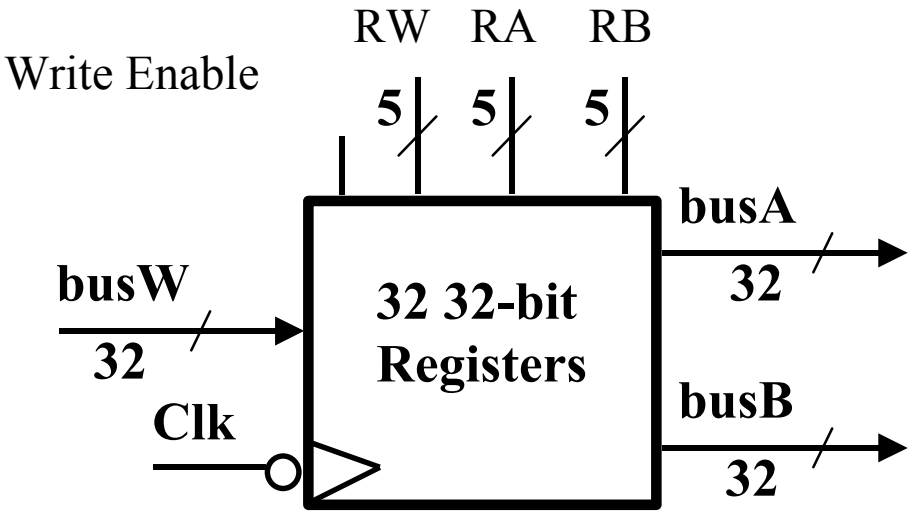
Clock input (C or CLK)

During read operation, register file behaves as a combinational logic block:

Register File

- Register File consists of 32 registers
- Two 32-bit output busses

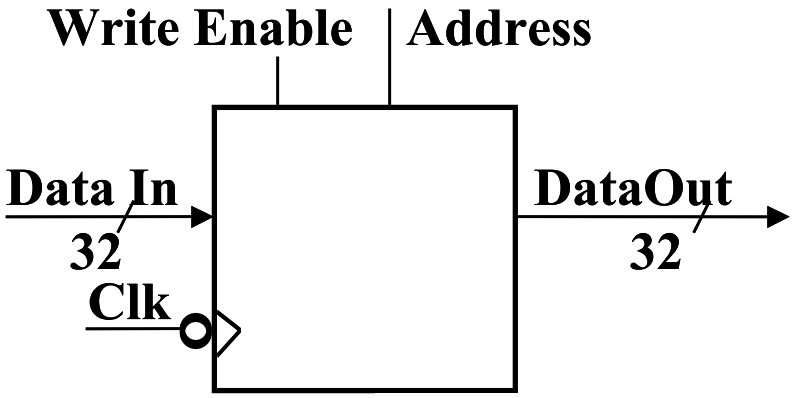
- One 32-bit input bus



- Register Read
  - RA (number) selects a register to put (data) on the busA
  - RB (number) selects the register to put on busB (data)
- Register Write
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1.

Ideal Memory

One input bus
One output bus



Memory word is selected by

- Address
- Write Enable = 1

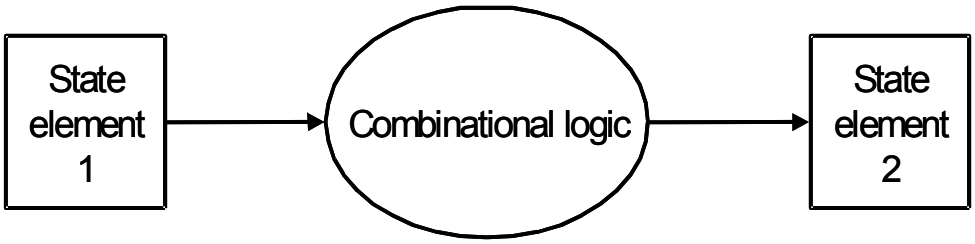
Clock input (CLK)

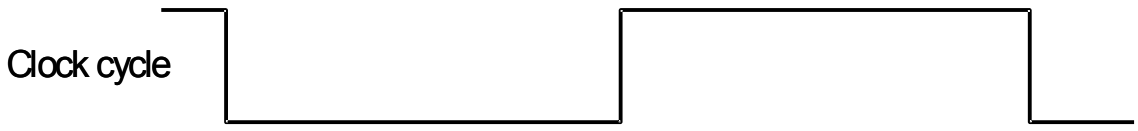
- CLK only required during write operation
- For read operation, memory behaves as a combinational logic.

Clocking Methodology

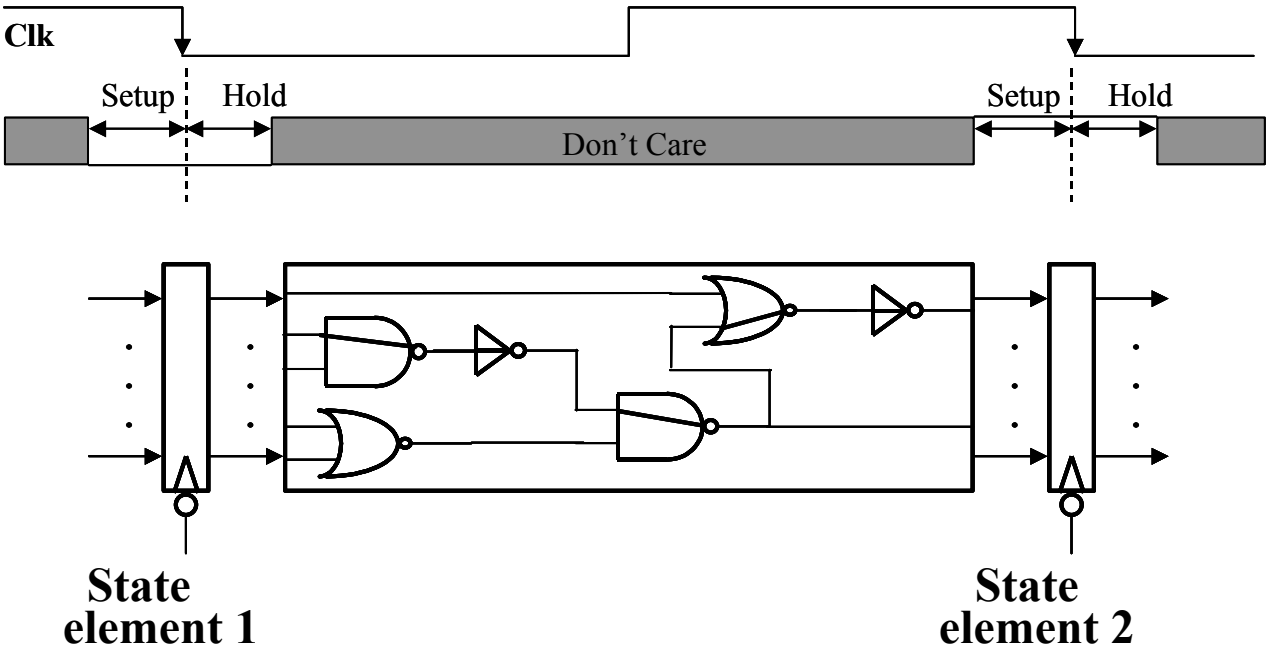
An edge triggered methodology
Typical execution:

- Read contents of some state elements.
- Send values through some combinational logic.
- Write results to one or more state elements





Clocking Methodology



Edge triggered clocking methodology

All storage elements are clocked by the same clock edge.

Cycle Time =
CLK-to-Q + Max-Delay-Path + Setup + Clock Skew

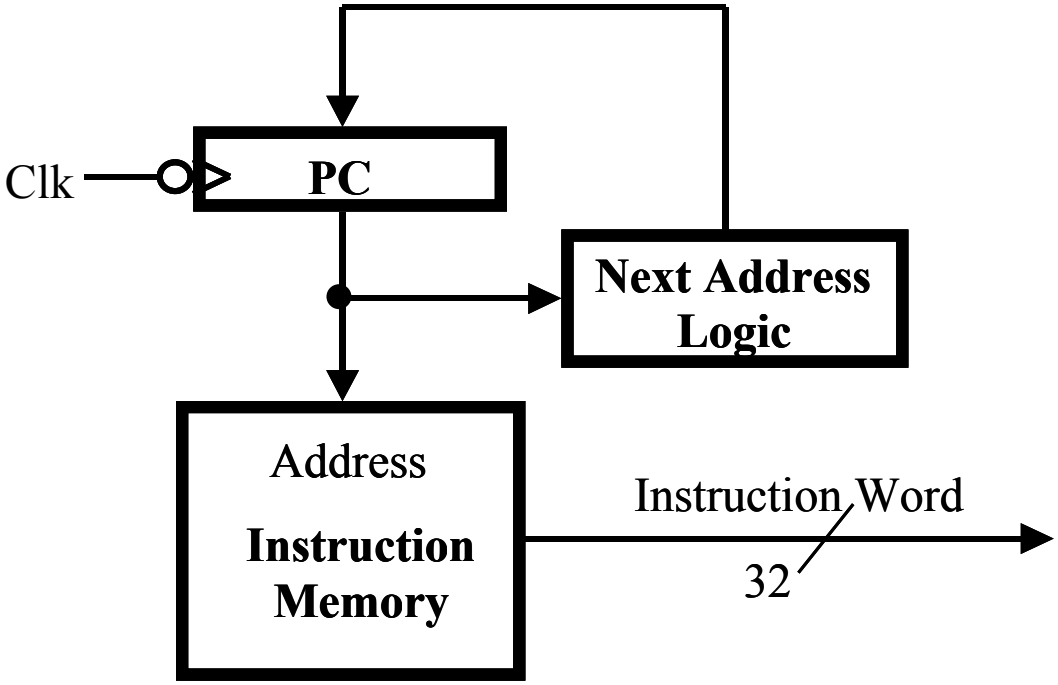
(CLK-to-Q + Min-Delay Path - Clock Skew)

Instruction Fetch Unit

Fetch the Instruction: mem[PC]

Update the program counter:

- Sequential Code
- Branch and Jump

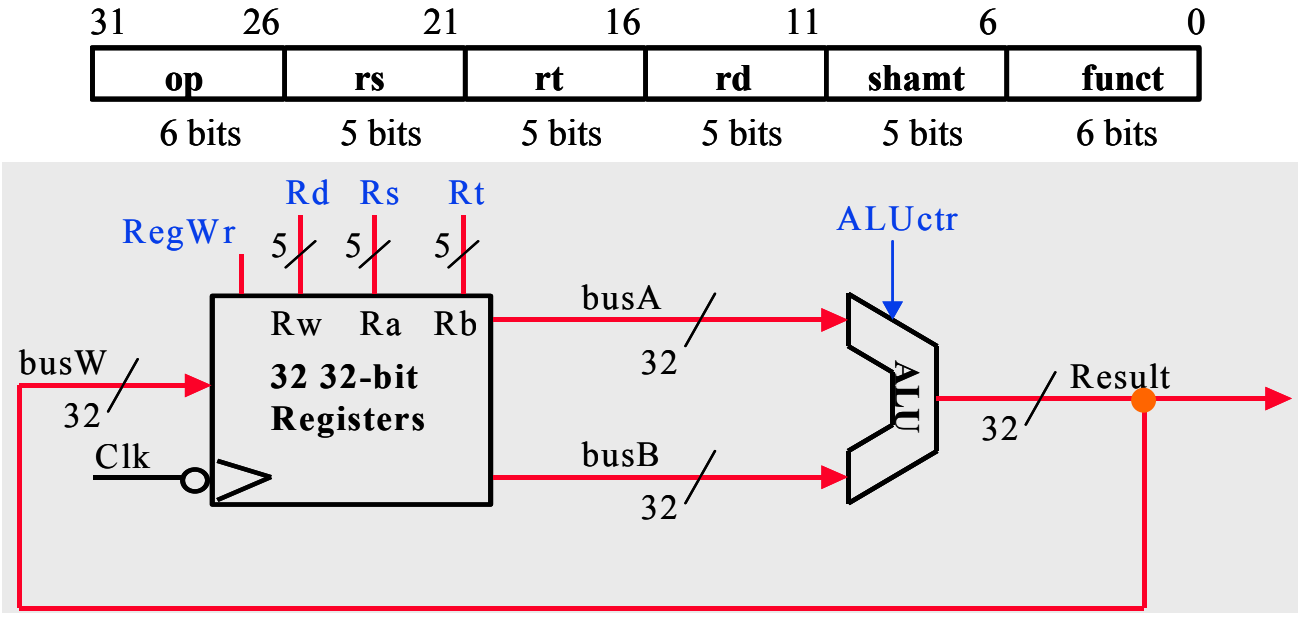


Add and Subtract

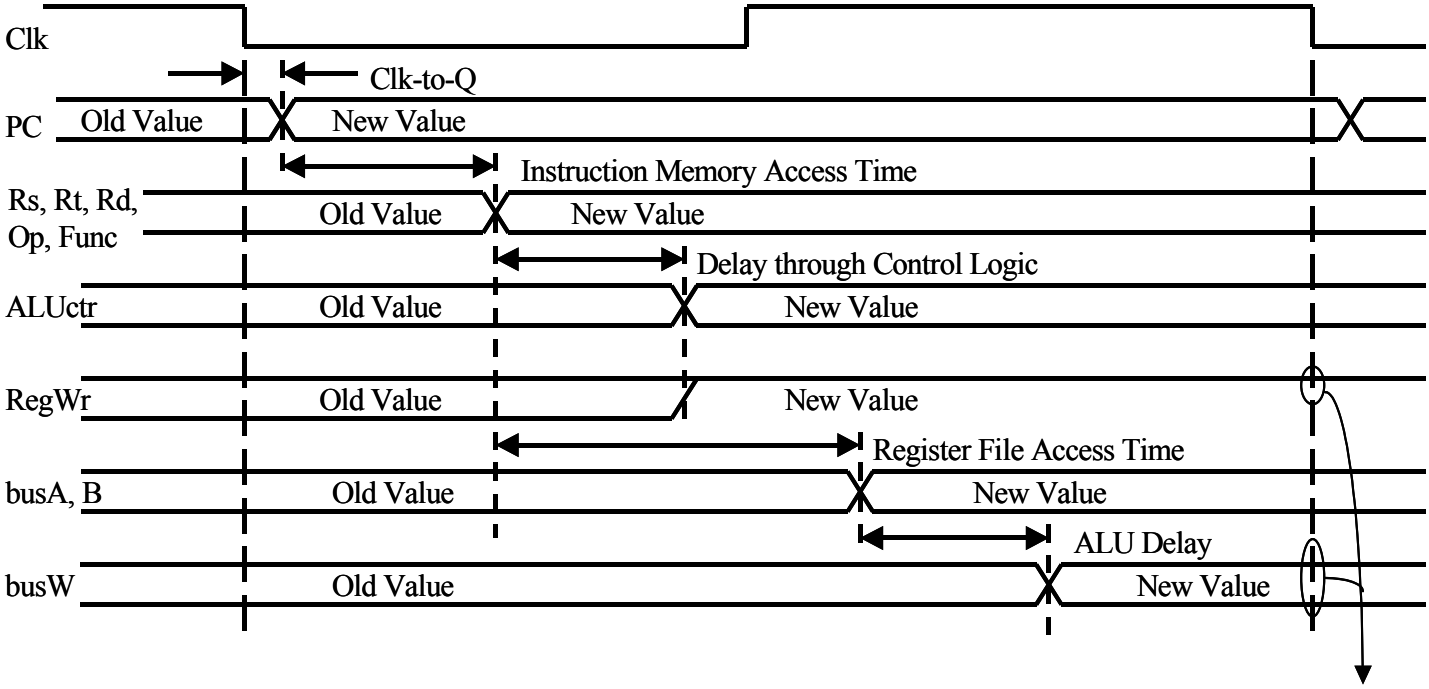
R[rd] <= R[rs] **op** R[rt]

For example: addU rd, rs, rt

- Ra, Rb and Rw come from rs, rt and rd fields
- ALUctr and RegWr



Register-Register Timing



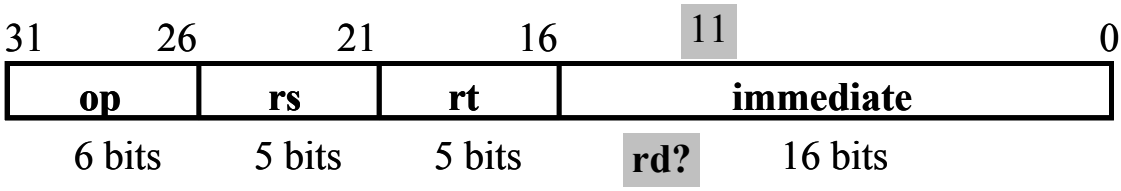
Two actions in parallel.

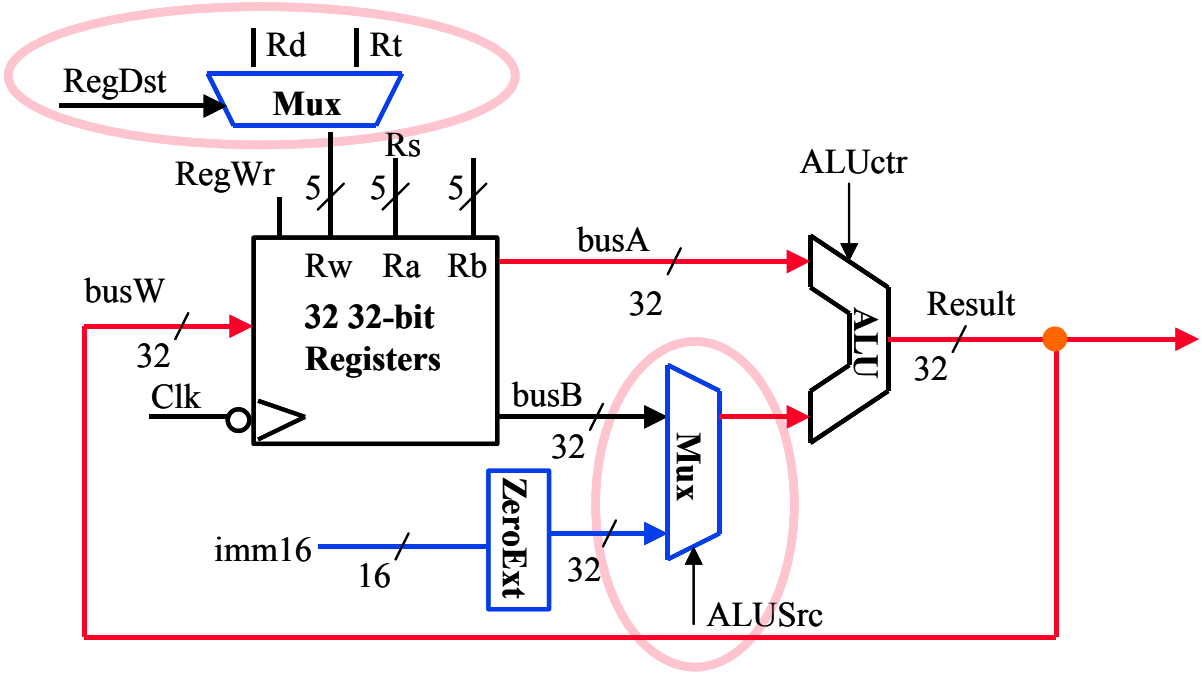
1. The control unit decode the Opcode and Func field and set the control signals ALUctr and RegWr.
2. While this is happening the register file is also read.

Logical Operations with Immediate

R[rt] <= R[rs] **OR** ZeroExt(imm16)

Example: **ori** rt, rs, imm16



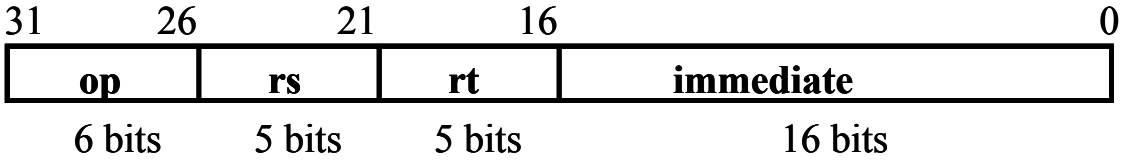


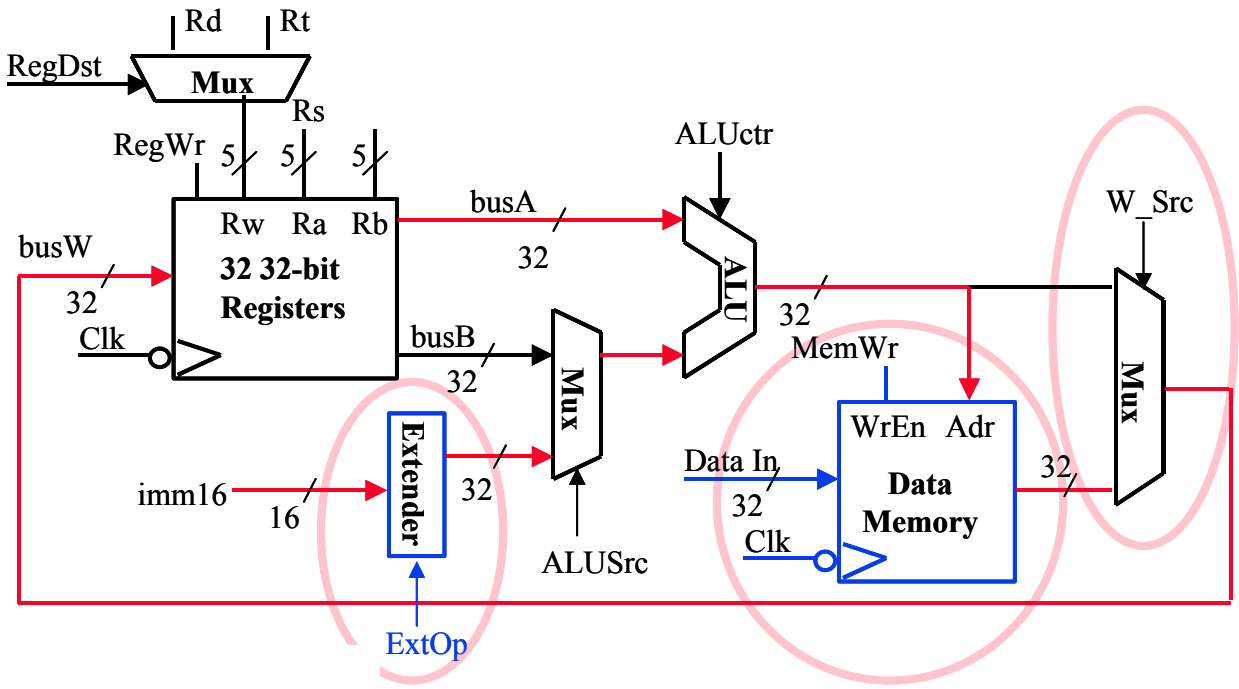
Two MUX are needed

Load Operations

R[rt] <= Mem[R[rs] + SignExt(imm16)]

Example: **lw** rt, rs, imm16

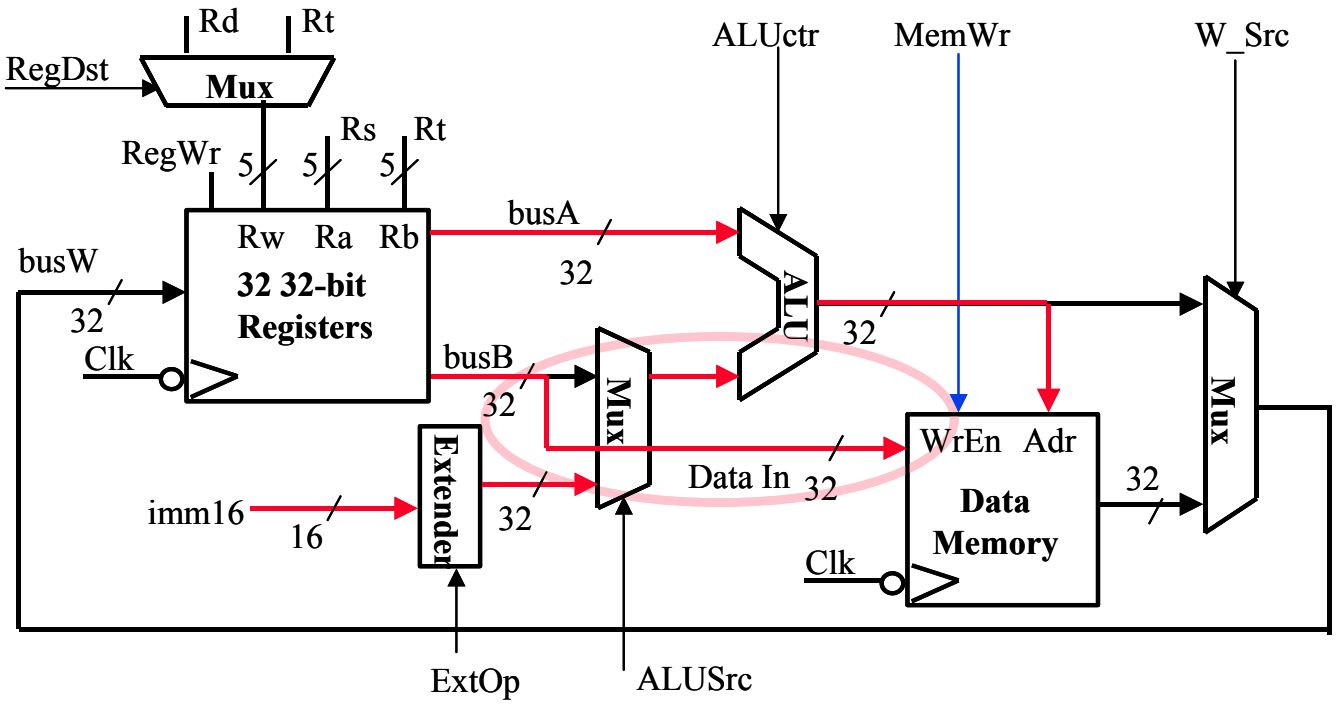




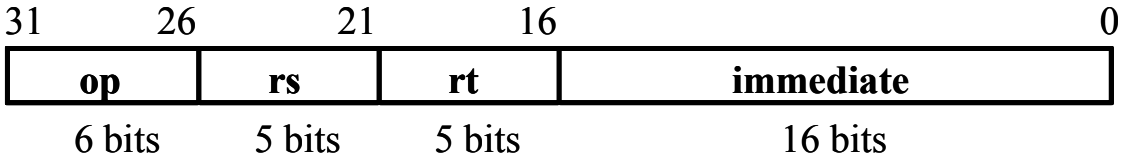
Store Operations

Mem[R[rs] + SignExt(imm16)] <= R[rt]

Example: **sw** rt, rs, imm16



The Branch Instruction



beq rs, rt, imm16

mem[PC] Fetch the instruction from memory

Equal <= (R[rs] = = R[rt]) Calculate branch condition

if (COND eq 0) Calculate the next instruction's address

then

PC <= PC + 4 + { SignExt(imm16), 2b"00 }

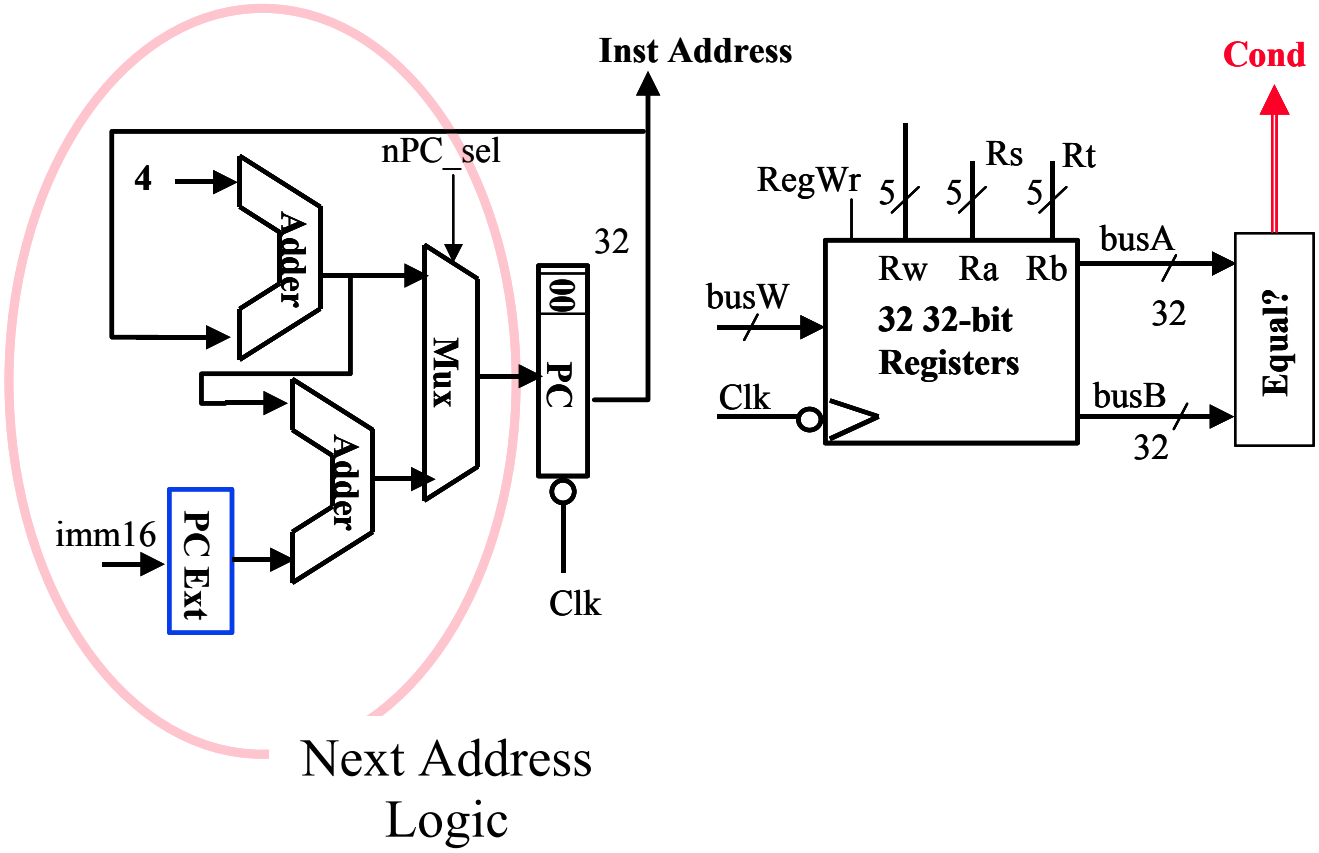
else

PC <= PC + 4

Datapath for Branch Operations

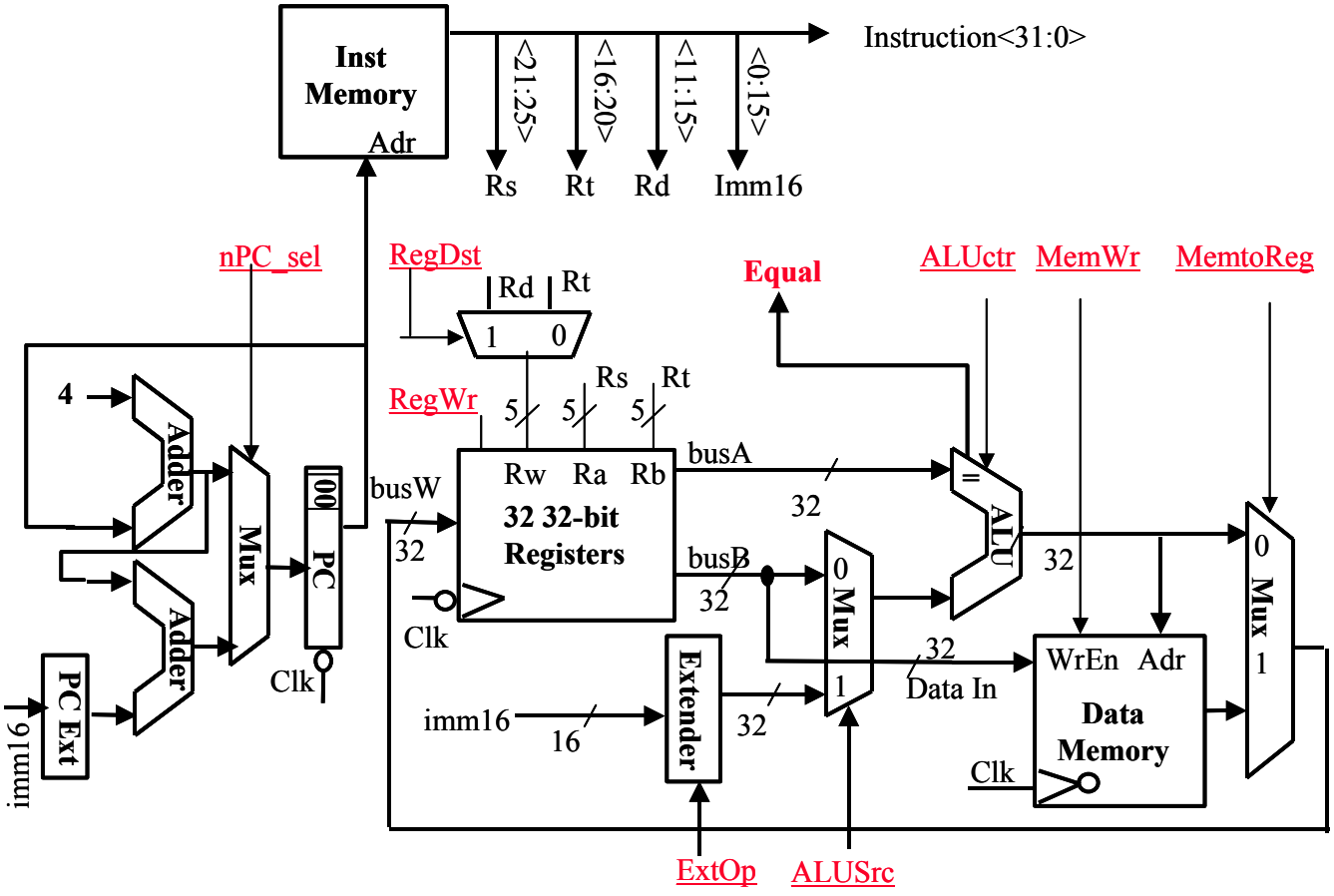
beq rs, rt, imm16

Datapath generates the (equal) condition



A Single Cycle Datapath

Putting it All Together

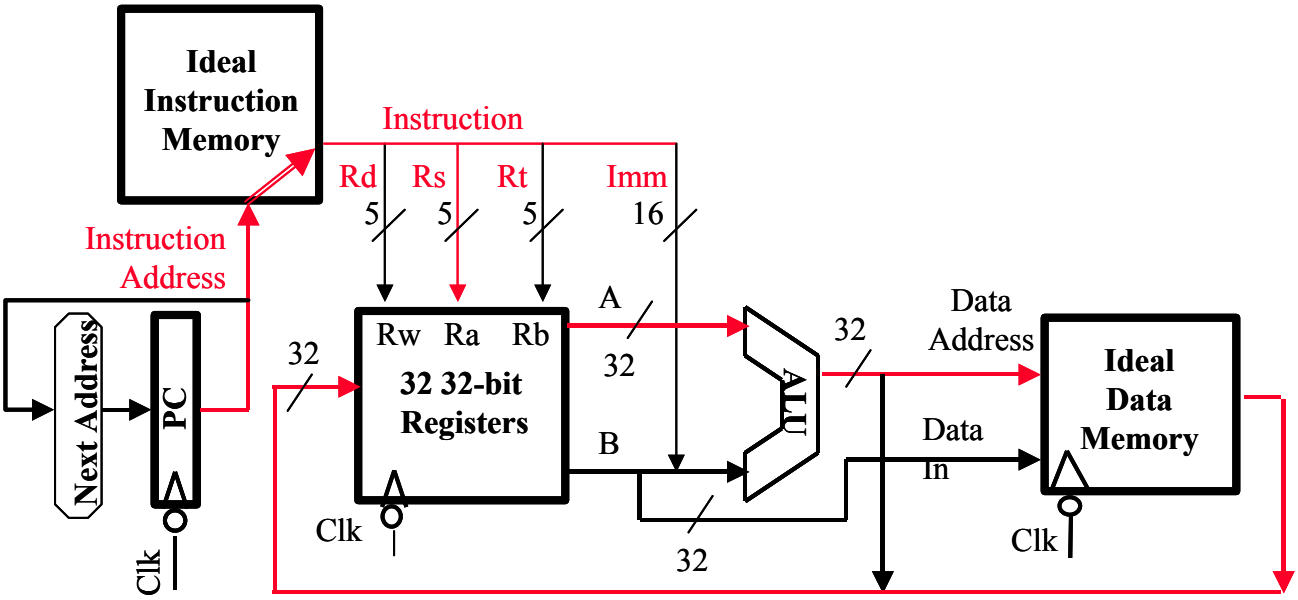


Abstract View of Critical Path

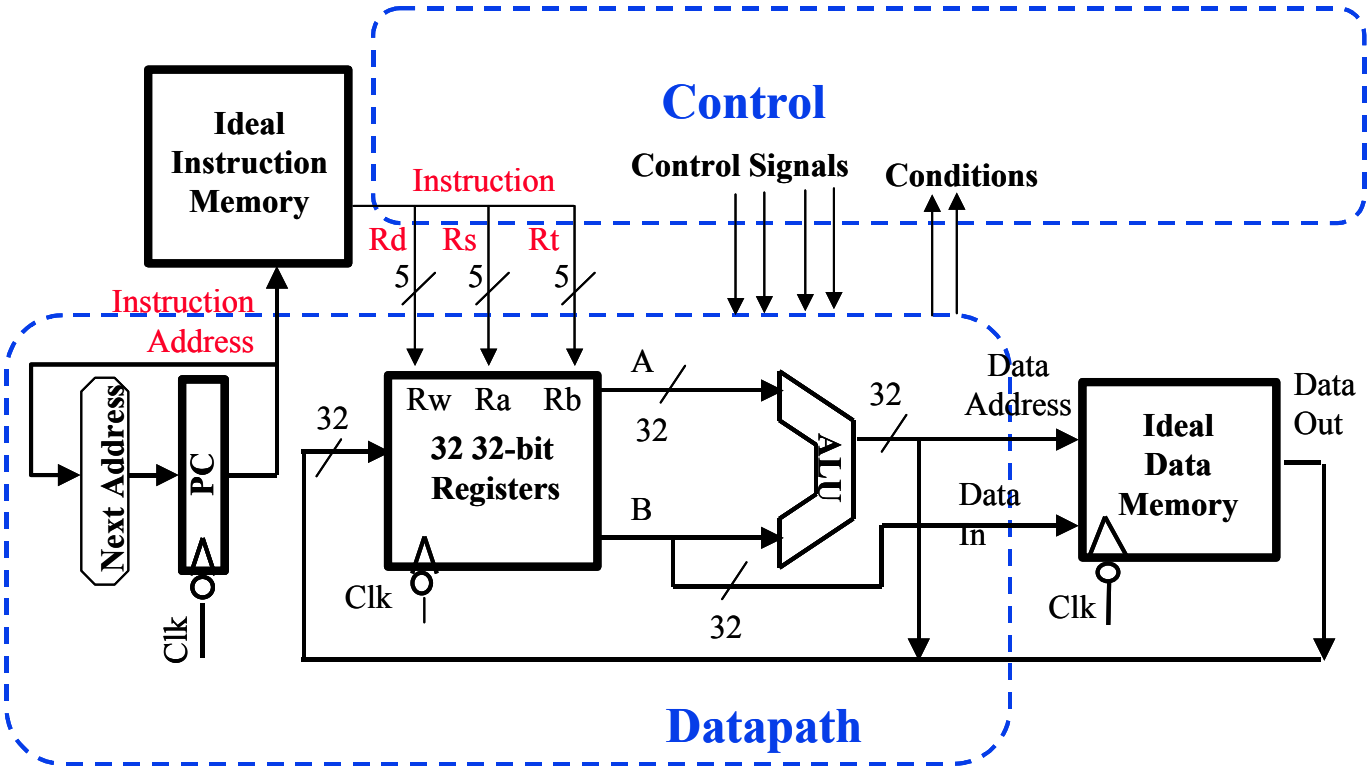
Register file and ideal memory:

During read operation:

Critical Path (Load Operation) =
PC's Clk-to-Q + Instruction Access Time +
Register File Access Time + ALU 32-bit Add +
Data Memory Access Time + Setup Time for Register File Write + Clock Skew



An Abstract View of the Implementation



Logical vs. Physical Structure

A Real MIPS Datapath