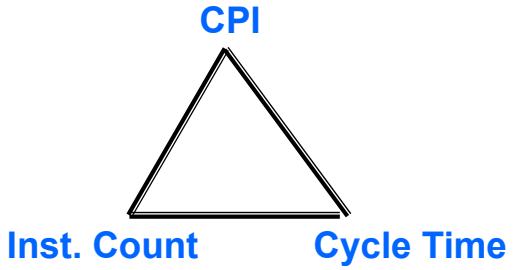
Chapter Four – I (1/4)

The Big Picture: The Performance Perspective

- Performance of a machine is determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction
- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Single cycle processor - one clock cycle per instruction
  - Advantages: Simple design, low CPI
  - Disadvantages: Long cycle time, which is limited by the slowest instruction.



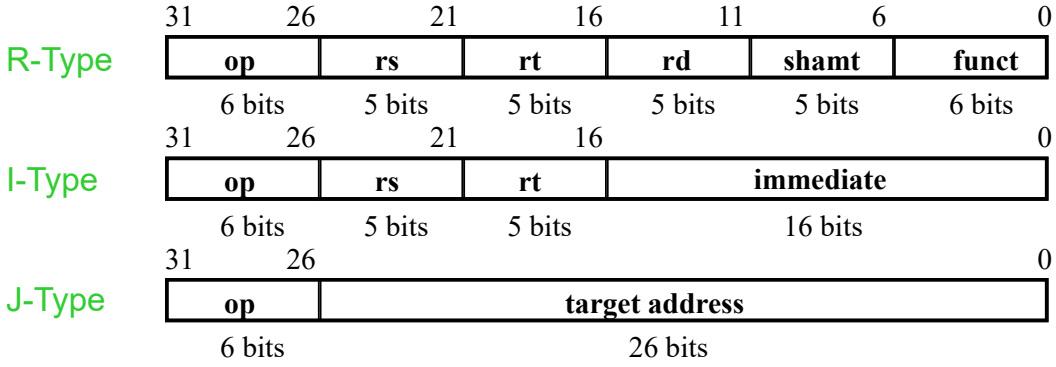
How to Design a Processor: step-by-step

1. Analyze instruction set → datapath requirements
   - the meaning of each instruction is given by register transfers

     R[rd] ← R[rs] + R[rt];
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Design datapath to meet the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Design the control logic

Review: The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats are:



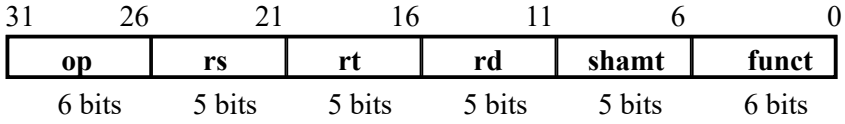
- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

The Processor: Datapath & Control

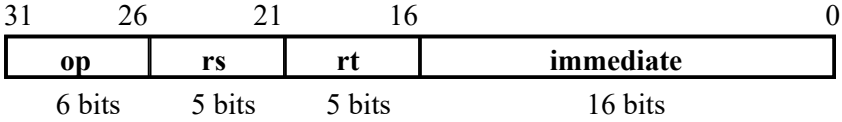
- Designing an implementation that includes a subset of the core MIPS instruction set:
  - The memory-reference instructions: `lw, sw`
  - The arithmetic-logical instructions: `add, sub, and, or, slt`
  - The control flow instructions: `beq, j`
- For every instruction except `j`, the first two steps are identical:
  - Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory
  - Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register.
- After these two steps, the actions required to complete the instruction depend on the instruction class
- All instructions use the ALU after reading the registers. Why?

Step 1a: The MIPS Subset

- R-Type
  - ADD and SUB
    - addu rd, rs, rt
    - subu rd, rs, rt



- I-Type
  - OR Immediate:
    - ori rt, rs, imm16
  - LOAD and STORE
    - lw rt, imm16(rs)
    - sw rt, imm16(rs)
  - BRANCH:
    - beq rs, rt, imm16



Register Transfer Logic (RTL)

- RTL gives the meaning of the instructions
- All instructions start by fetching the instruction

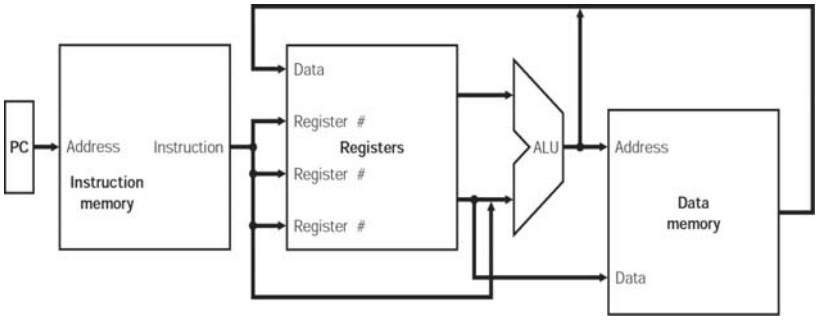
```
op | rs | rt | rd | shamt | funct = MEM[ PC ]
op | rs | rt |   Imm16             = MEM[ PC ]

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)<<2)
              else PC ← PC + 4
```


Step 1: Requirements of the Instruction Set

- Memory
  - instruction & data
- Registers (32 x 32)
  - read rs
  - read rt
  - write rt or rd
- PC
- Extender (sign extend or zero extend)
- Add and sub register or extended immediate
- Add 4 or shifted extended immediate to PC

More Implementation Details

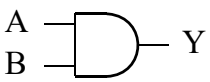
- **Abstract / Simplified View:**



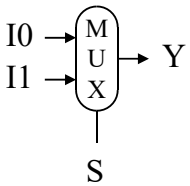
- **The functional units in the MIPS implementation consist of two different types of logic elements:**
  - **Elements that operate on data values (combinational)**
    - Outputs depend only on the current inputs.
  - **Elements that contain state (sequential)**
    - They have internal storage.
    - At least two inputs (clock and data) and one output (data)
    - E.g. D flip-flop

Step 2: Components of the Datapath

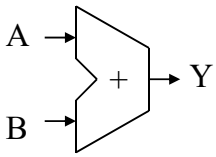
- **Combinational Elements: Does not use a clock**
  - AND-gate
    - Y = A & B



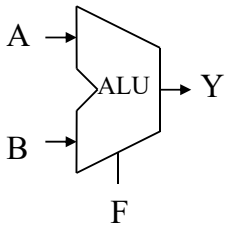
  - Multiplexer
    - Y = S ? I1 : I0



  - Adder
    - Y = A + B

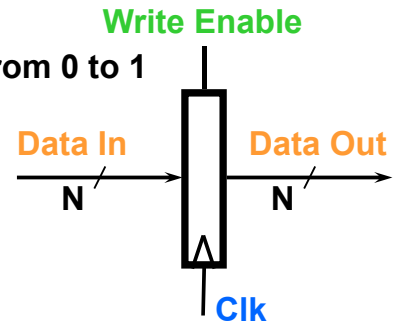


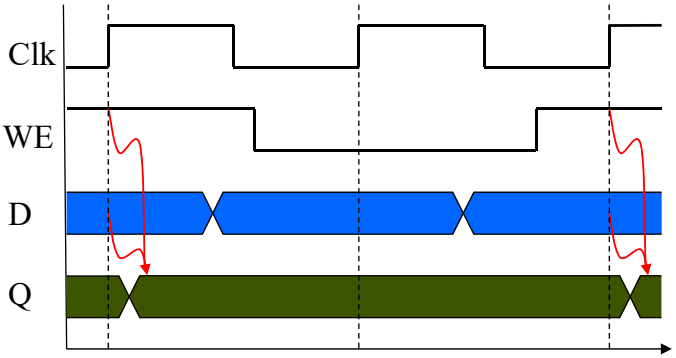
  - Arithmetic/Logic Unit
    - Y = *f*(A, B)

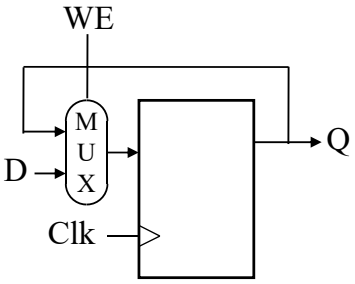


Sequential Element

- **Register (Basic Building Blocks)**
  - **Edge-triggered: update when Clk changes from 0 to 1**
  - **Similar to the D Flip Flop except**
    - 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 on the falling edge of the clock

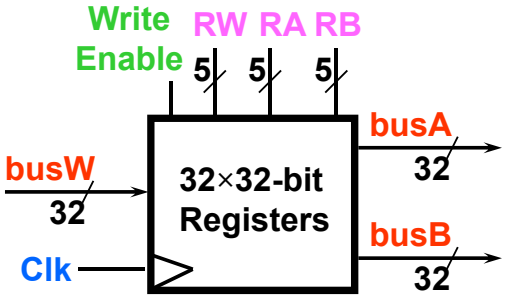






Storage Element: Register File

- **Register File consists of 32 registers:**
  - **Two 32-bit output busses: busA and busB**
  - **One 32-bit input bus: busW**

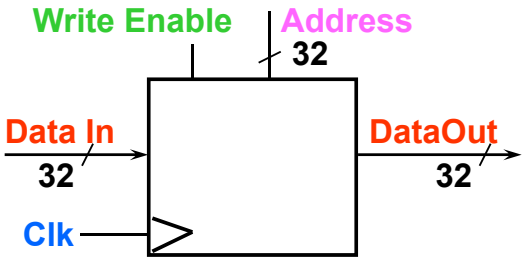


- **Register is selected by:**
  - **RA (number) selects the register to put on busA (data)**
  - **RB (number) selects the register to put on busB (data)**
  - **RW (number) selects the register to be written via busW (data) when Write Enable is 1**

- **Clock input (CLK)**
  - **The CLK input is a factor ONLY during write operation**
  - **During read operation, behaves as a combinational logic block:**
    - RA or RB valid => busA or busB valid after “access time.”

Storage Element: Idealized Memory

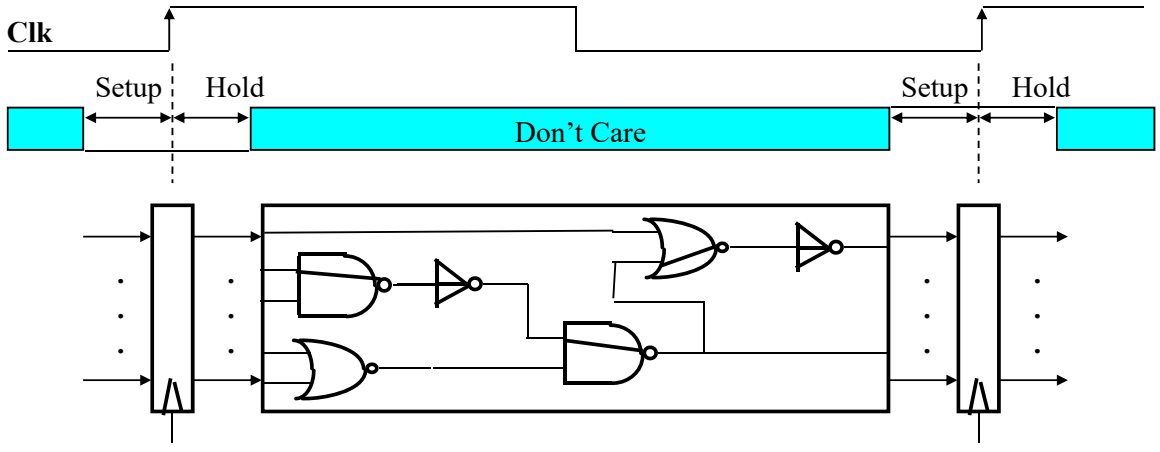
- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out



- Memory word is selected by:
  - 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

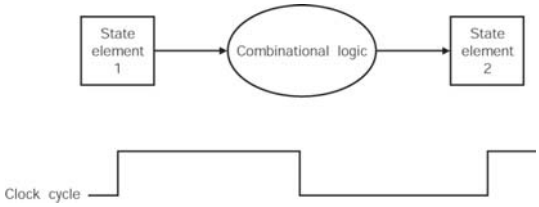
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, memory behaves as a combinational logic block:
    - Address valid => Data Out valid after “access time.”

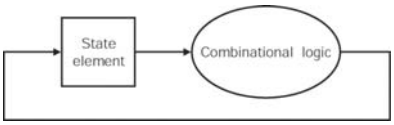
Clocking Methodology – Positive Edge Triggered



- All storage elements are clocked by the same clock edge
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
- (CLK-to-Q + Shortest Delay Path – Clock Skew) > Hold Time

Combinational vs. Sequential





- **Combinational logic, state elements, and the clock**
  - **Any inputs to a state elements must reach a stable value before the active clock edge causes the sate to be updated.**
  - **The time necessary for the signals to travel from SE1 to SE2 defines the clock cycle length**

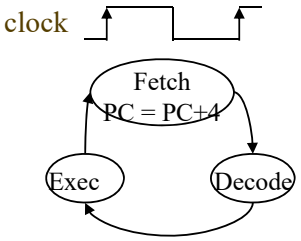
- **Allows a state element to be read and written in the same clock cycle without causing a race condition.**

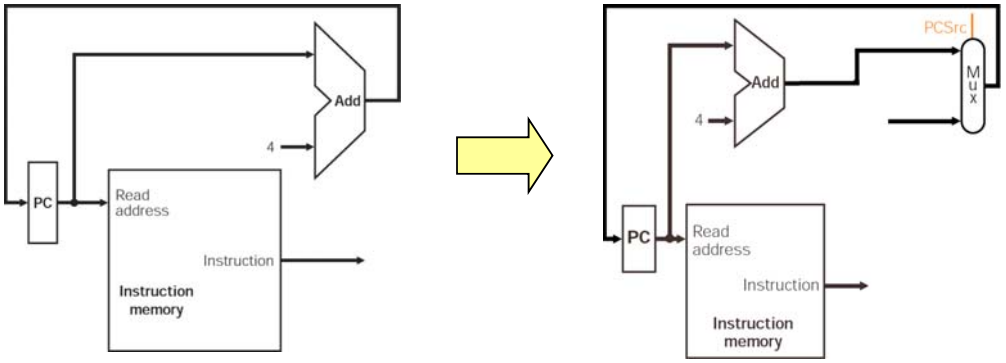
Step 3: Design Datapath

- **Register Transfer Requirements → Datapath Design**
  - **Instruction Fetch**
  - **Decode instructions and Read Operands**
  - **Execute Operation**
  - **Write back the result**

Datapath for Instruction Fetch

- **The common RTL operations**
  - **Fetch the Instruction: mem[PC]**
  - **Update the program counter:**
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← "something else"

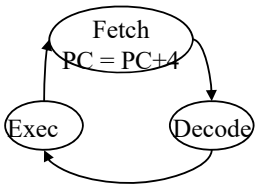


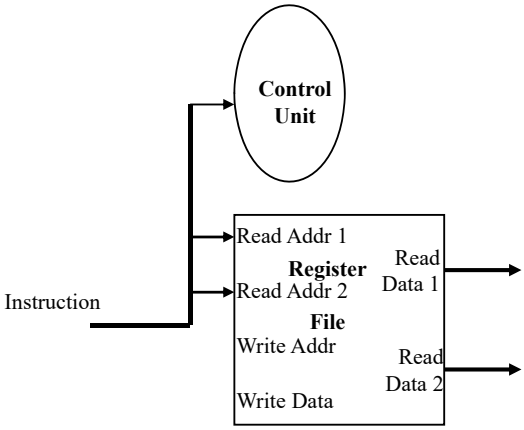


  - **PC is updated every clock cycle, so it does not need an explicit write control signal just a clock signal**
  - **Reading from the Instruction Memory is a combinational activity, so it doesn't need an explicit read control signal**

Decoding Instructions

- **Decoding instructions involves**
  - **sending the fetched instruction's opcode and function field bits to the control unit**

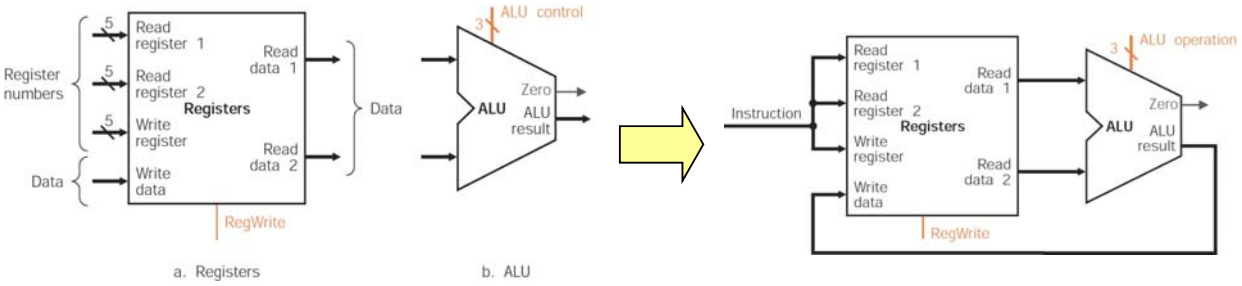


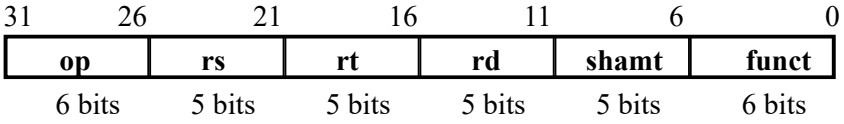


  - **And**
  - **Reading two values from the Register File**
    - Register File addresses are contained in the instruction

Datapath for R-type ALU operations

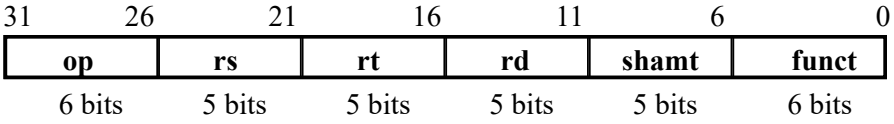
- **Register file and ALU**

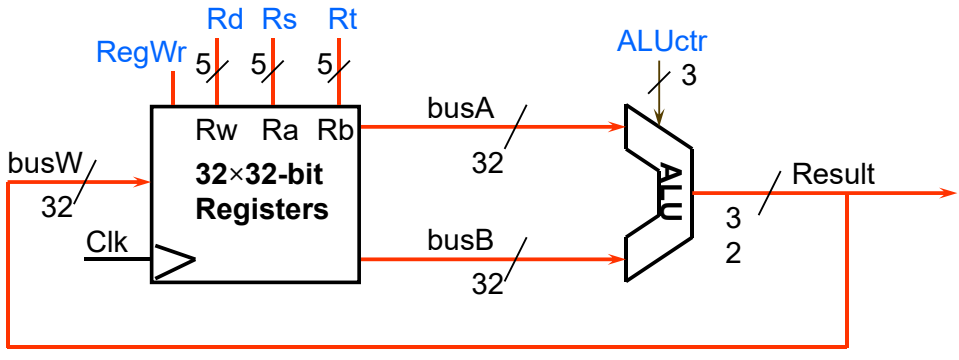




Datapath for Add & Subtract

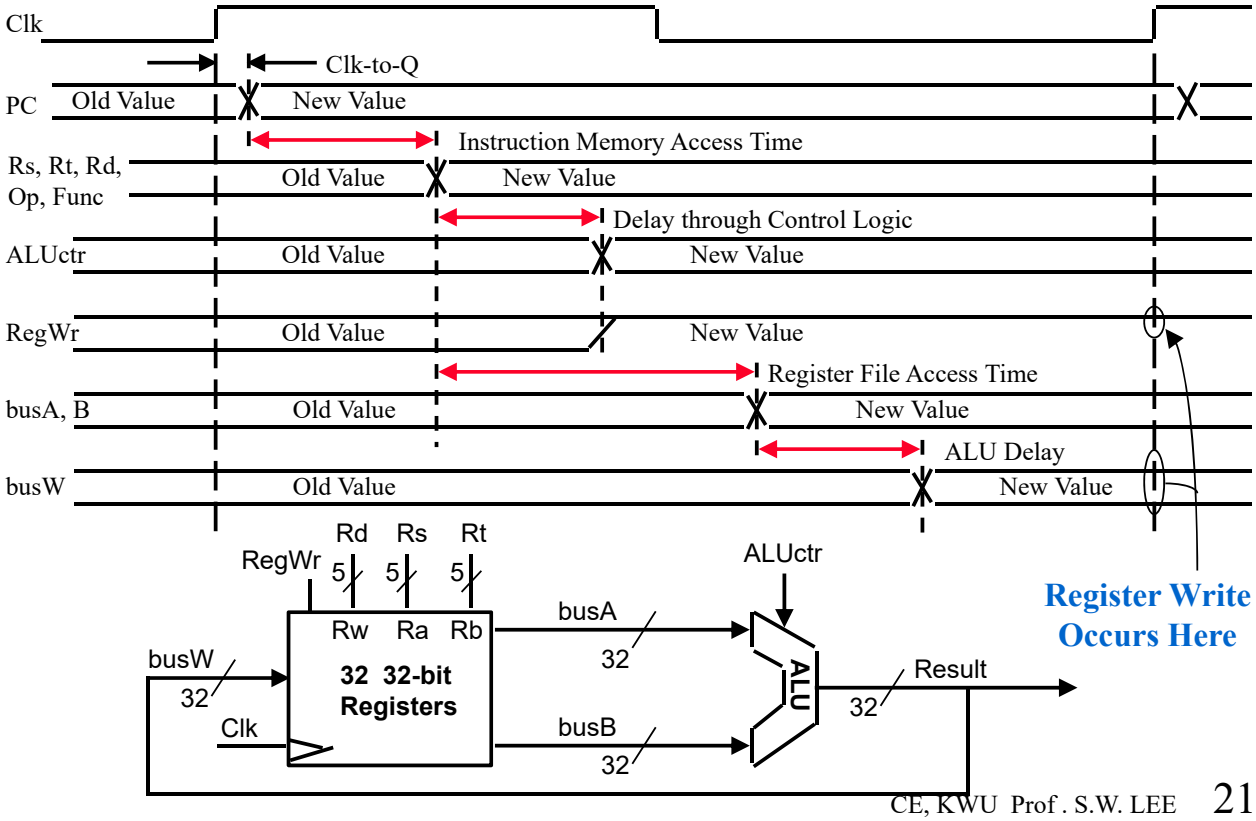
- **R[rd] ← R[rs] op R[rt] Example: addu rd, rs, rt**
  - **Ra, Rb, and Rw come from instruction's rs, rt, and rd fields**
  - **ALUctr and RegWr: control logic after decoding the instruction**





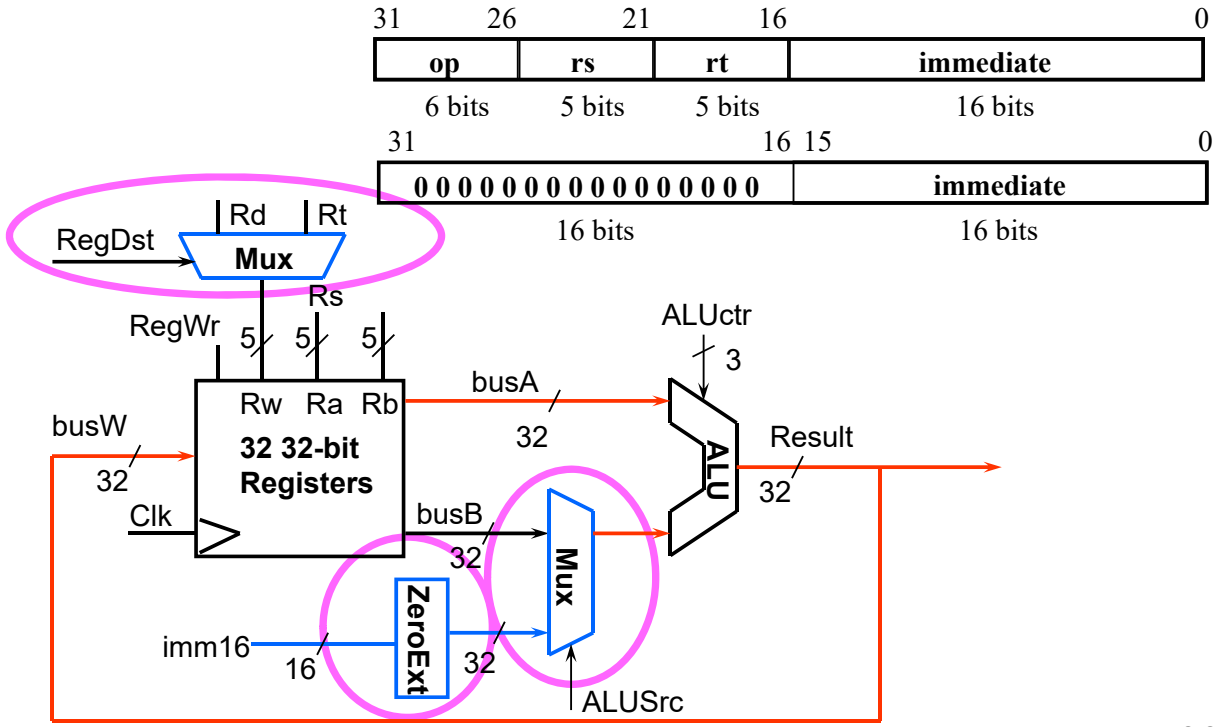
  - **Note that Register File is not written every cycle (e.g. sw), so we need an explicit write control signal for the Register File**

Register-Register Timing: One complete cycle

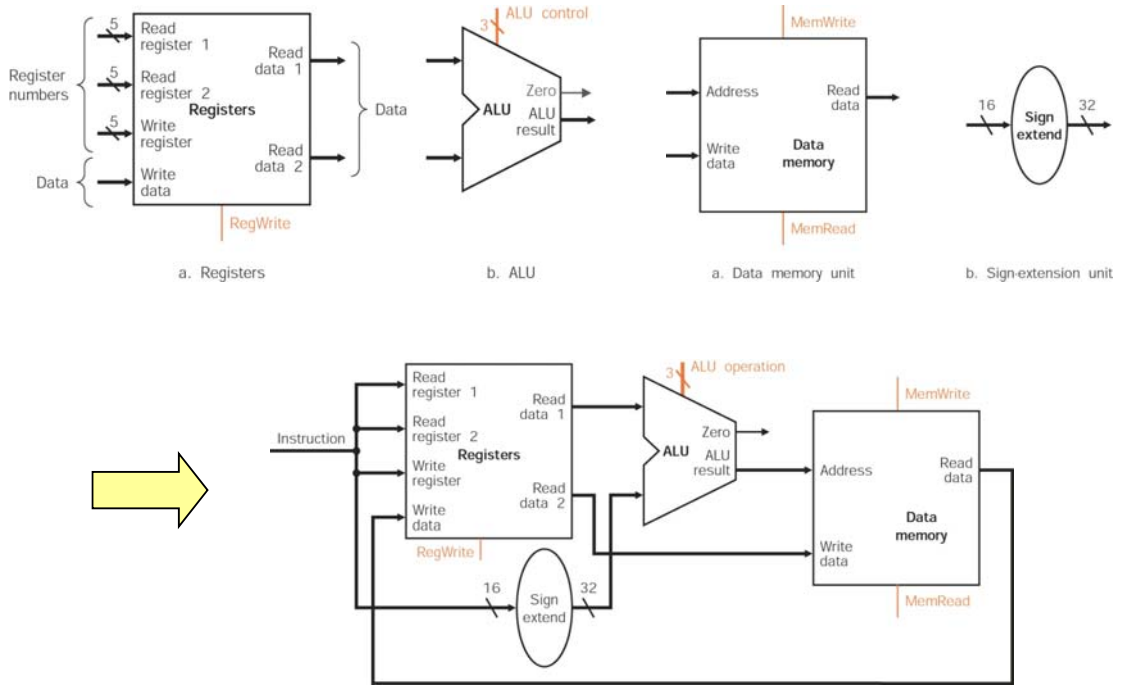


Logical Operations with Immediate

- **R[rt] ← R[rs] op ZeroExt[imm16] Example : ori rt, rs, imm16**

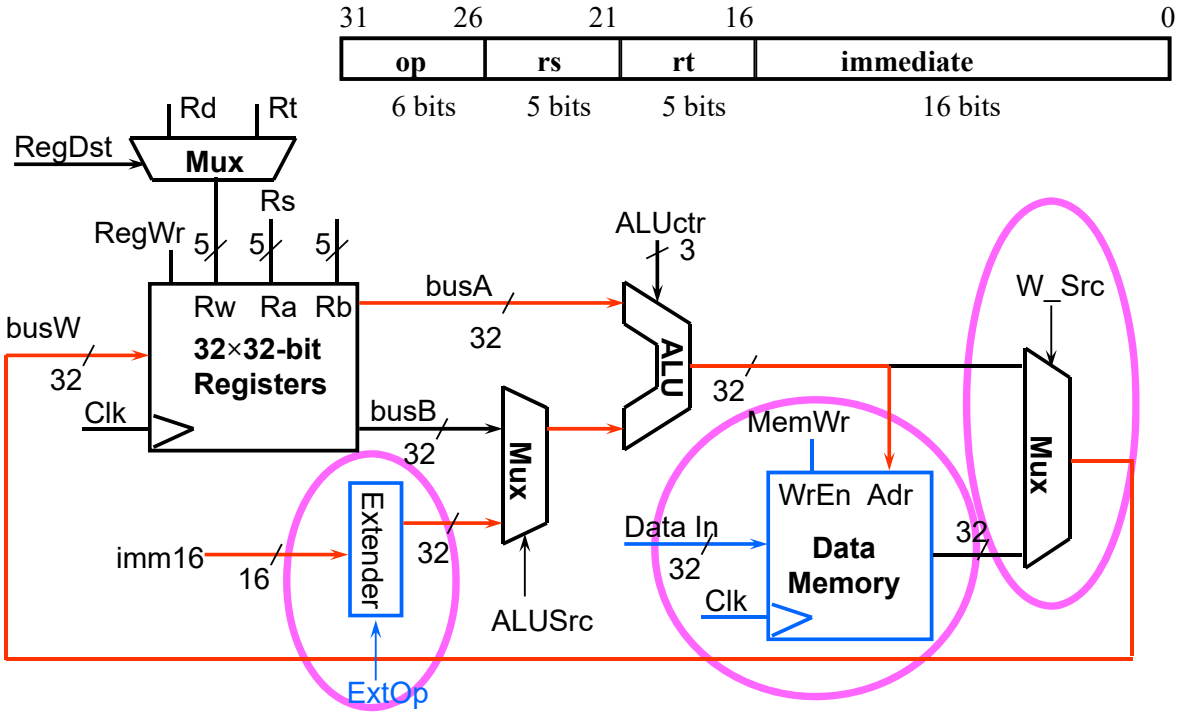


Datapath for Loads and Stores



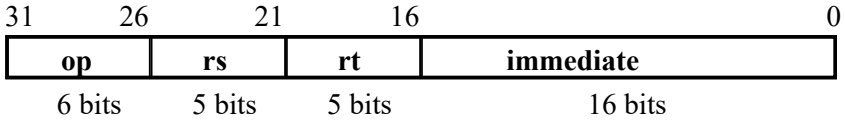
Load Operations

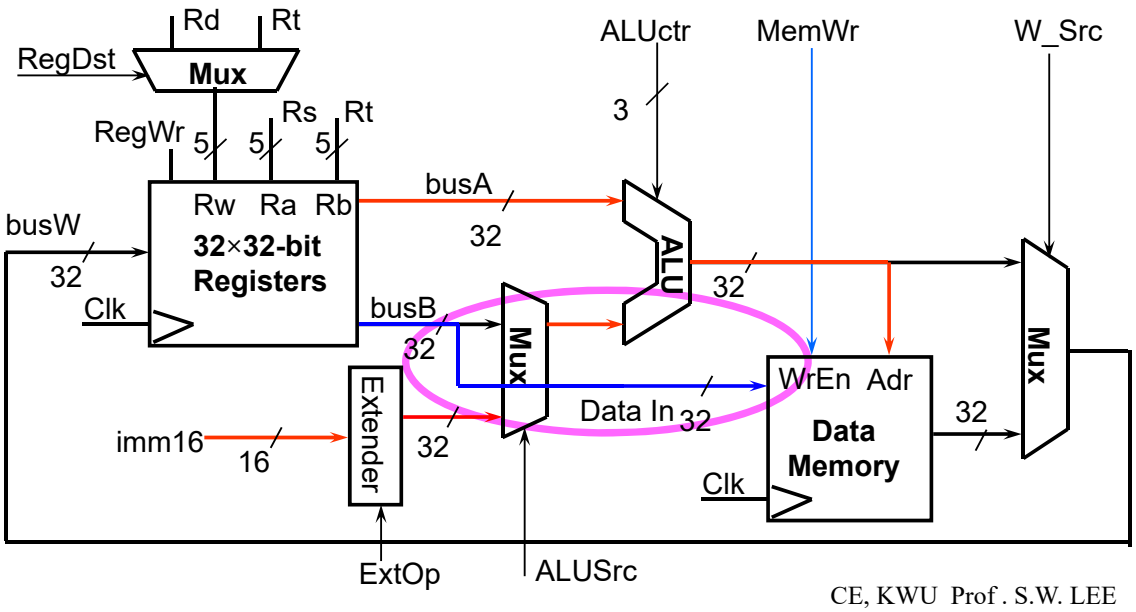
- **R[rt] ← Mem[R[rs] + SignExt[imm16]** **Example: lw rt, rs, imm16**



Store Operations

- **Mem[R[rs] + SignExt[imm16] ← R[rt]] Example: sw rt, rs, imm16**





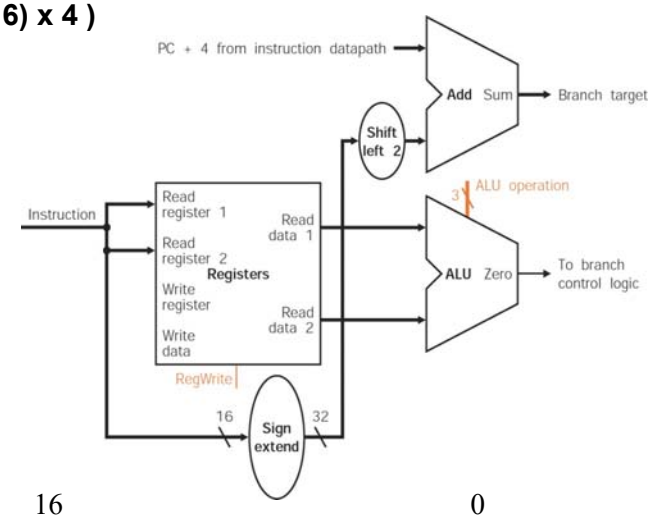
Datapath for Branch

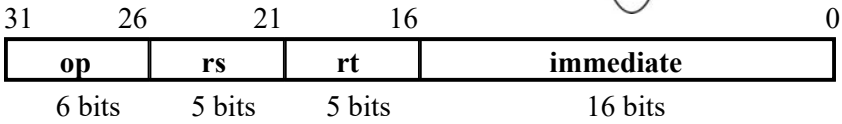
- **beq rs, rt, imm16**
  - **mem[PC]** **Fetch the instruction from memory**
  - **Equal = R[rs] == R[rt]** **Calculate the branch condition**
  - **if (COND eq 0)** **Calculate the next instruction's address**

    PC ← PC + 4 + (SignExt(imm16) x 4)

    else

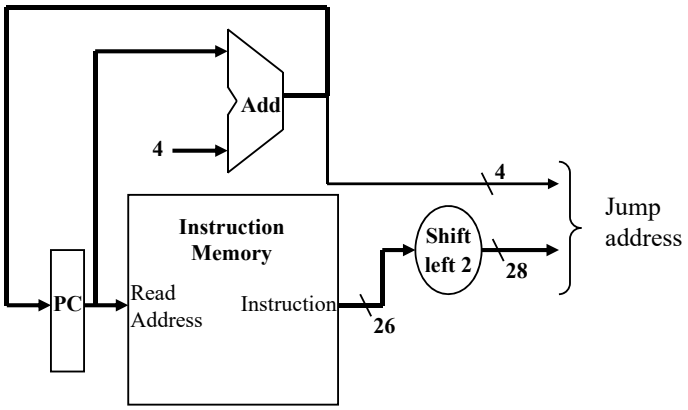
    PC ← PC + 4

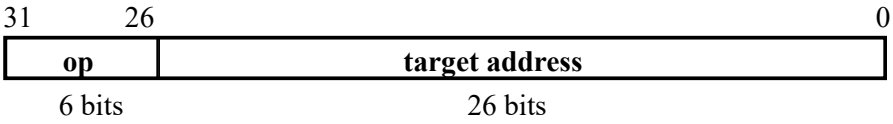




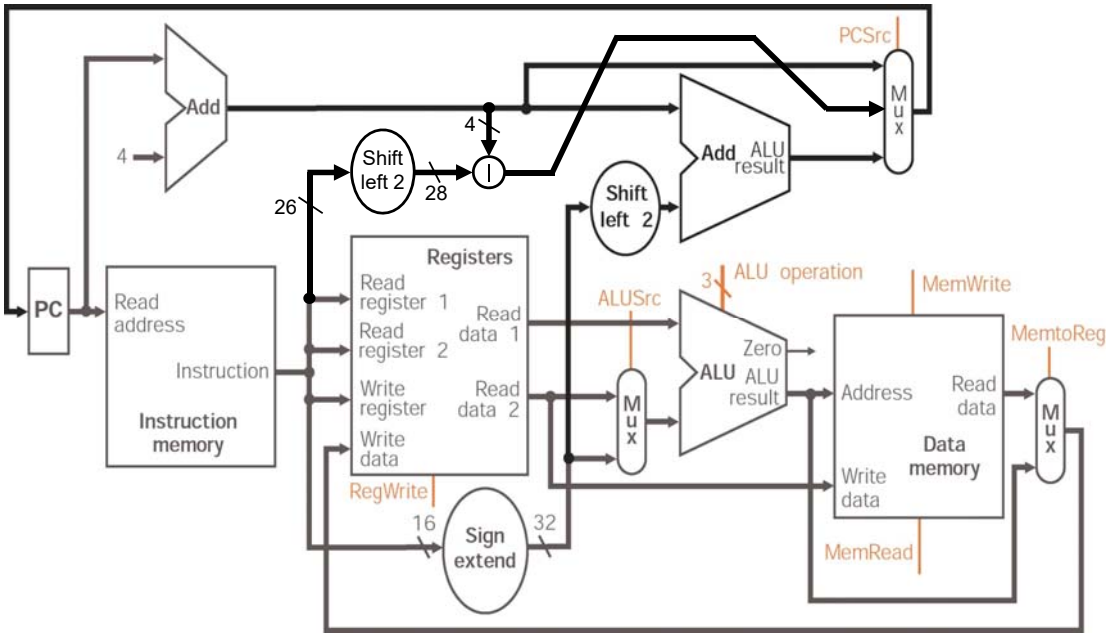
Executing Jump Operations

- **Jump operation involves**
  - **replace the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits**





Putting it All Together: A Single Cycle Datapath



Creating a Single Datapath from the Parts

- **Assemble the datapath segments and add control lines and multiplexors as needed**
- **Single cycle design – fetch, decode and execute each instructions in one clock cycle**
  - **no datapath resource can be used more than once per instruction, so some must be duplicated (e.g., separate Instruction Memory and Data Memory, several adders)**
  - **multiplexors needed at the input of shared elements with control lines to do the selection**
  - **write signals to control writing to the Register File and Data Memory**
- **Cycle time is determined by length of the longest path**