# Single tapath

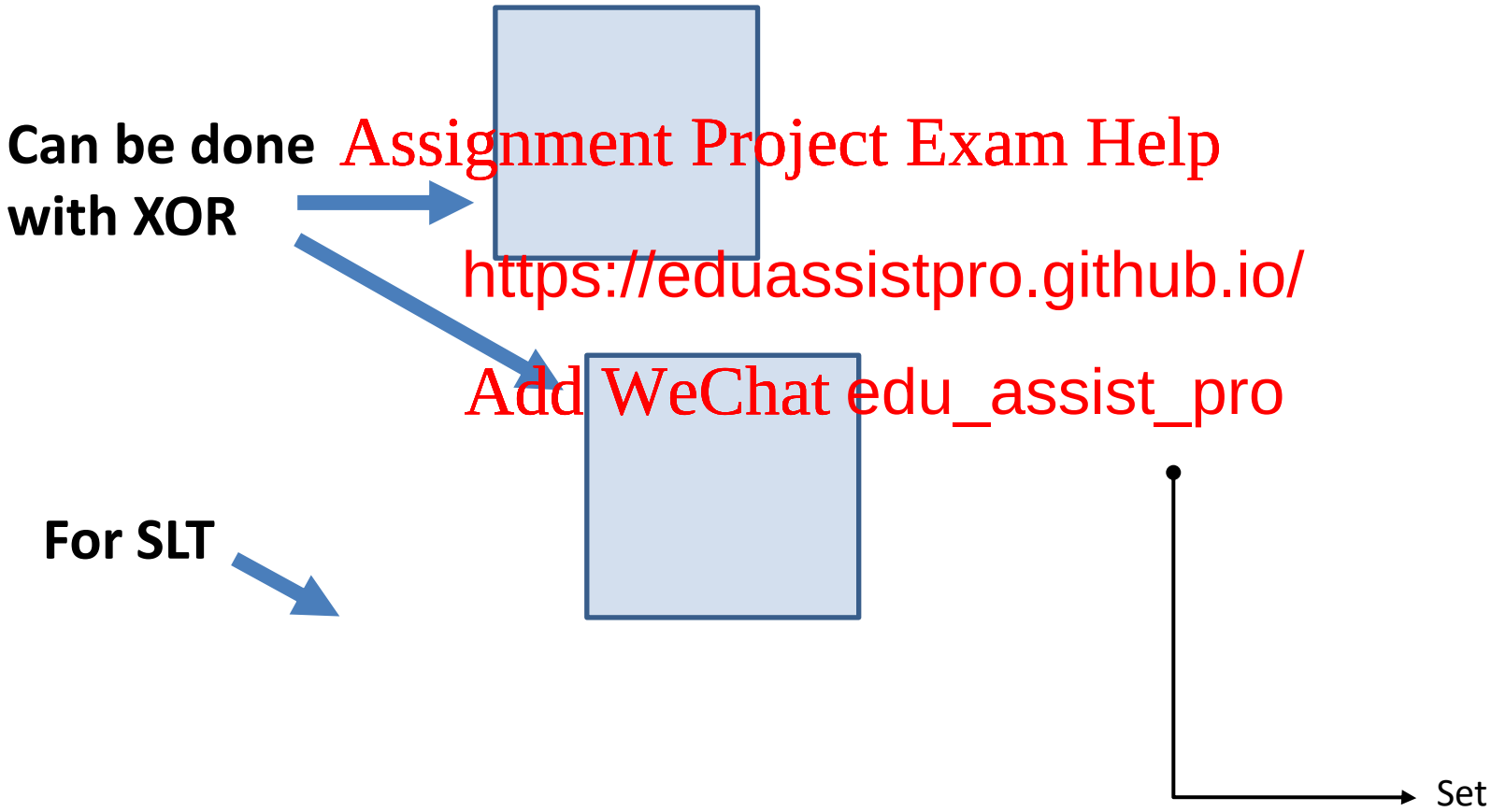


COMP

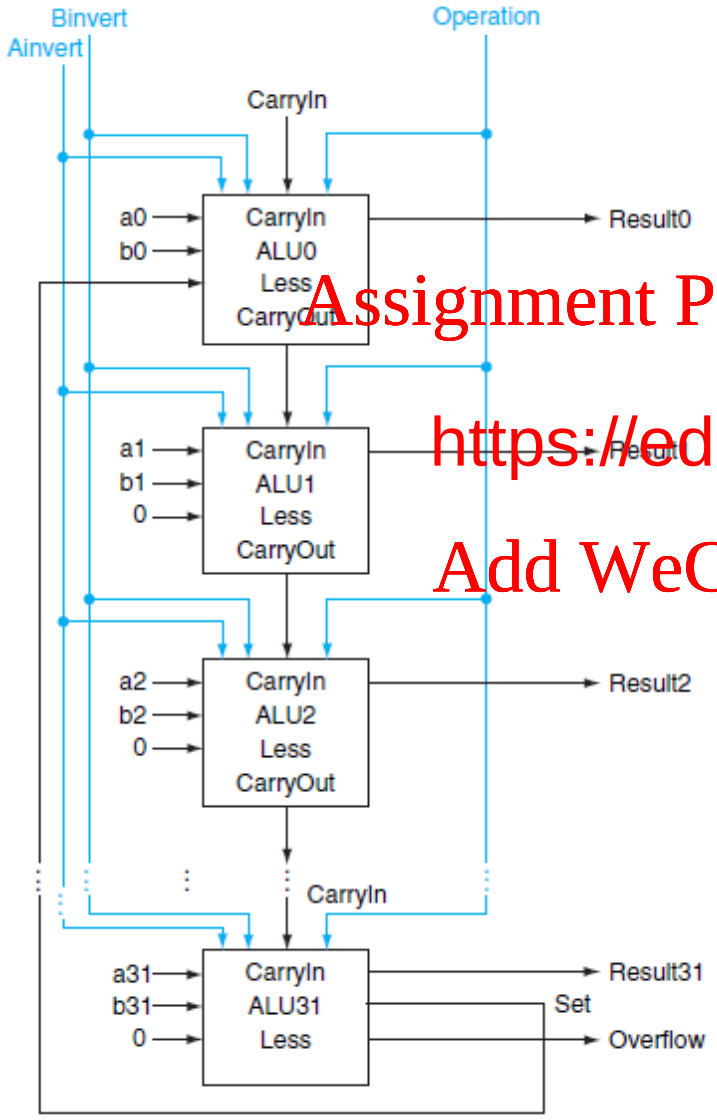
# Review from earlier in the term

- Use multiplexer (mux) to select among input
  - S input bits selects $2^S$ inputs
  - Each input can be n-bits wide, independent of S
- ALU can be impleme
  - Coupled with basic b
- N-bit adder-subtractor done usi – adders with XOR gates on input
  - XOR serves as conditional inverter
- Programmable Logic Arrays are often used to implement our Control Logic (for instance, in a finite state machine)

# Review, 1 bit ALU



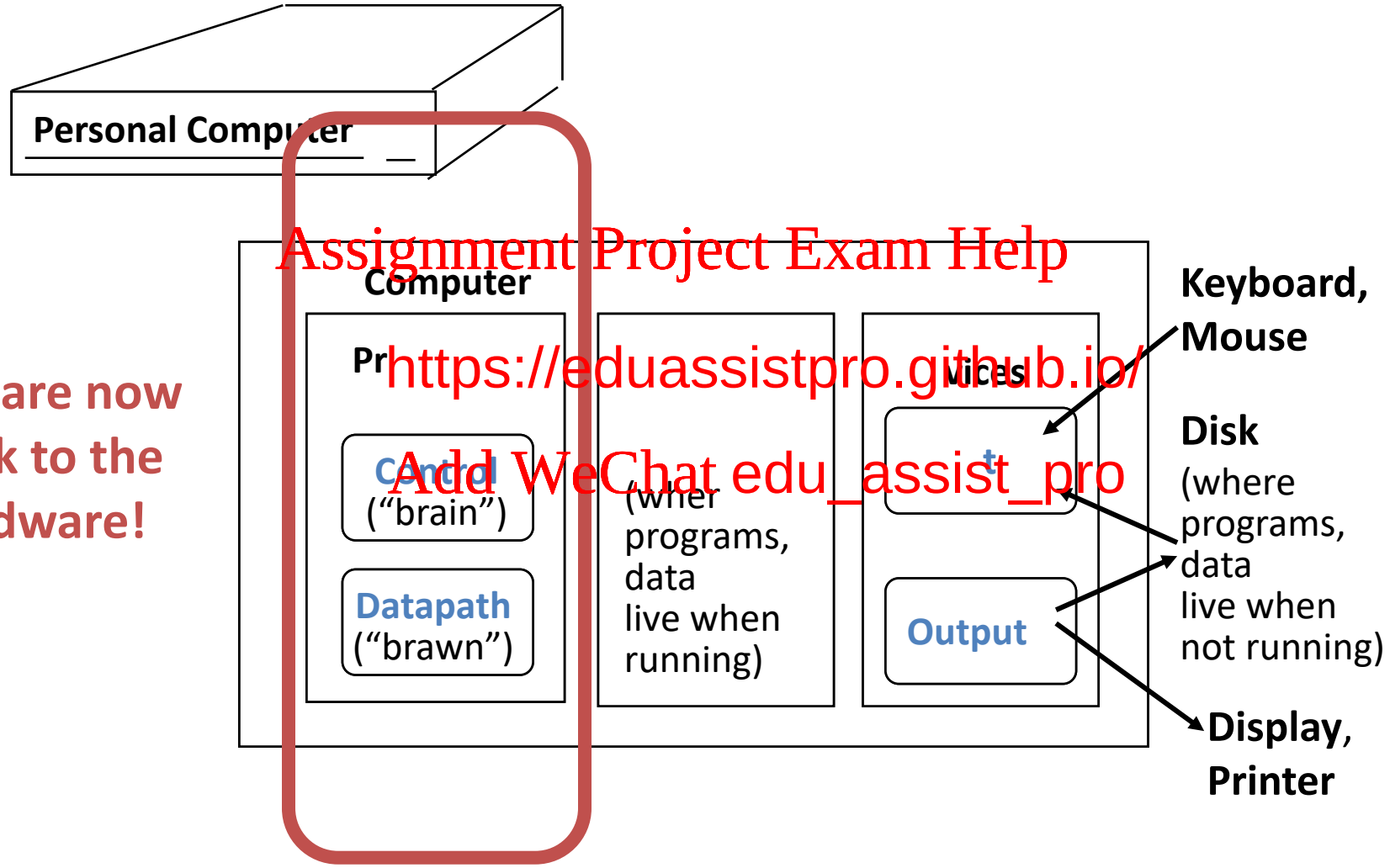
# Review, 32 bit ALU






| Instruction | Ainvert | Binvert | CarryIn | Operation |
|---|---|---|---|---|
| | | 0 | 0 | 2 |
| | | 1 | 1 | 2 |
| AND | 0 | 0 | x | 0 |
| OR | 0 | 0 | x | 1 |
| NOR | 1 | 1 | x | 0 |
| SLT | 0 | 1 | 1 | 3 |

# Review: 5 parts of any Computer




We are now back to the hardware!

# Outline

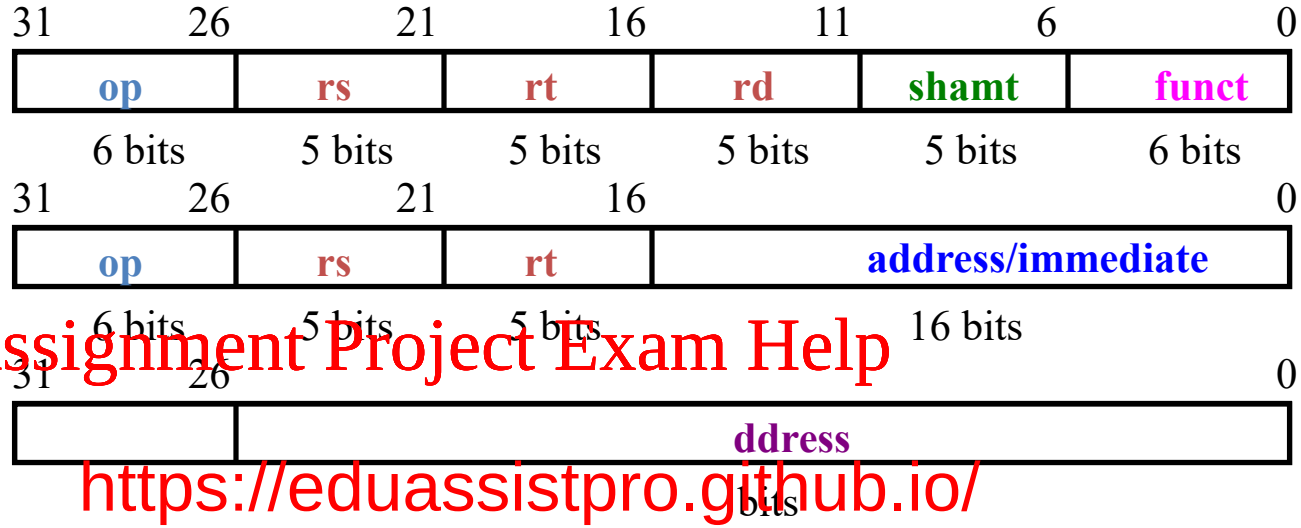
- Design a processor: step-by-step
- Requirements of the Instruction Set
- Hardware compone nstruction set requirements

# How to Design a Processor: step-by-step

1. Analyze instruction set architecture (ISA) ⇒ datapath requirements
   - Meaning of each instruction is given by the *register transfers*
   - Datapath must include storage element for ISA registers
   - Datapath must support
2. Select set of datapath components a h clocking methodology
3. Assemble datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic

# Review: The MIPS Instruction Formats

- All MIPS instructions are 32 bits long, 3 formats:
  - R-type
  - I-type
  - J-type

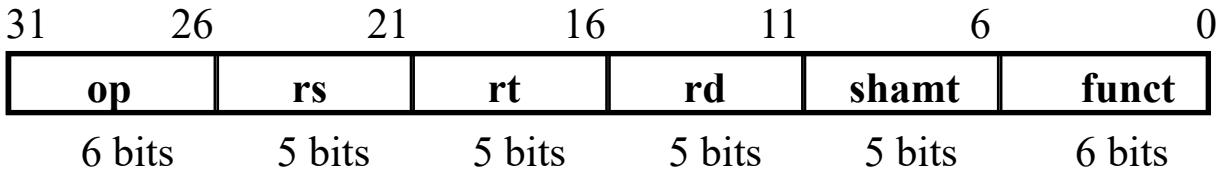


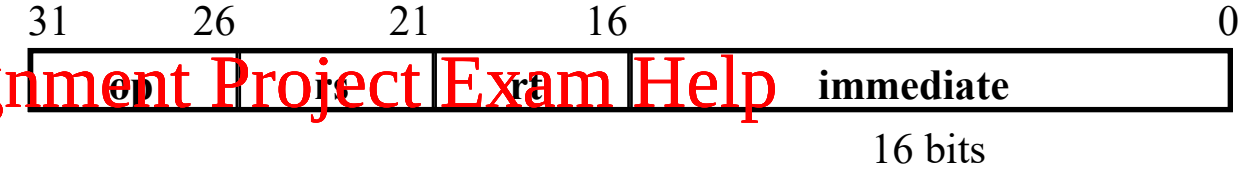


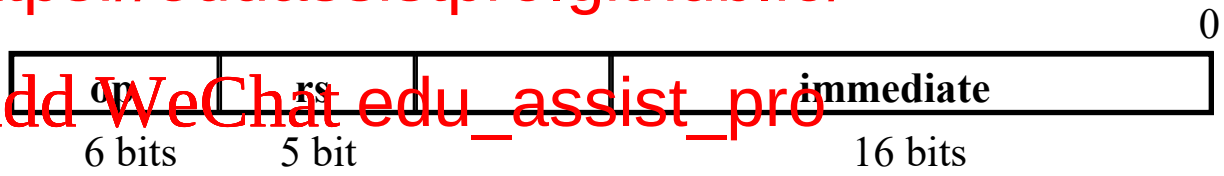

- The different fields are:
  - op: operation ("opcode") of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: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

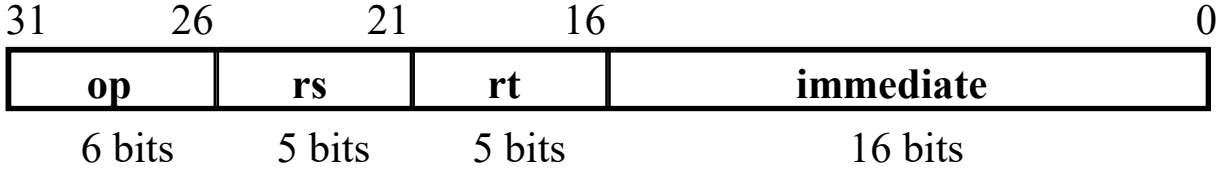
# Step 1a: The MIPS-lite Subset for today

- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`
- OR Immediate:
  - `ori rt,rs,imm`
- LOAD and STORE Word
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`
- BRANCH:
  - `beq rs,rt,imm16`



# Register Transfer Language

- RTL gives the ***meaning*** of the instructions
  - {op , rs , rt , rd , shamt , funct} = MEM[PC]
  - {op , rs , rt , Imm16} = MEM[PC]
- Start by fetching the instruction, then execute transfers



| Instruction | R |
|---|---|
| ADDU | R 4 |
| SUBU | R[rd] = R[rs] – R[rt] |
| ORI | R[rt] = R[rs] \| zero 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<br>PC = PC + 4 + (sign_ext(Imm16) \|\| 00)<br>else PC = PC + 4 |

# Step 1: Requirements of the Instruction Set

- Memory (MEM)
  - instructions & data
- Registers (R: 32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC
- Extender (sign extend)
- Add and Sub: register or extended immediate
- Add 4 or extended immediate to PC

# Step 2: Components of the Datapath

- Combinational Elements
- Storage Elements
  - Clocking methodology
    - We will use falling e
      see a small circle in front of the cloc

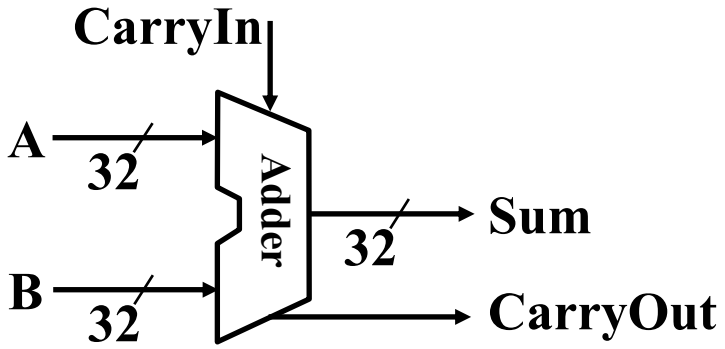


hese examples, thus you will

Logisim puts a negation circle in front of the clock to denote falling edge trigger. The circle is absent if you have a rising edge trigger.

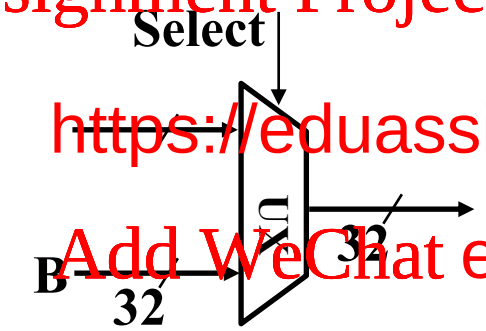
# Combinational Logic Elements (Building Blocks)

- Adder

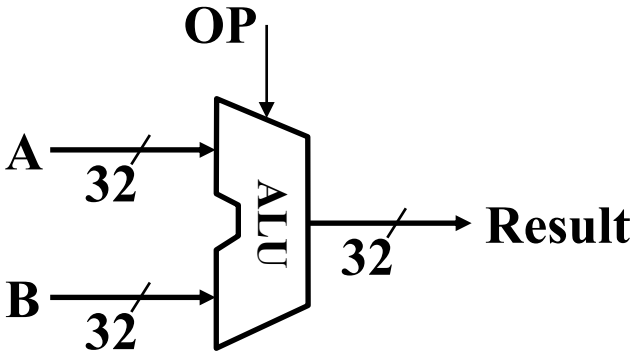





- MUX




- ALU



# ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==

```
ADDU  R[rd] = R[rs] + R[rt]; ...
SUBU  R[rd] = R[rs] - R[rt]; ...
ORI   R[rt] = R[rs] | xt(Imm16)...
BEQ   if ( R[rs] == R[
```



- Test to see if output == 0 LU operation lets us implement the == equality test. ***How?***

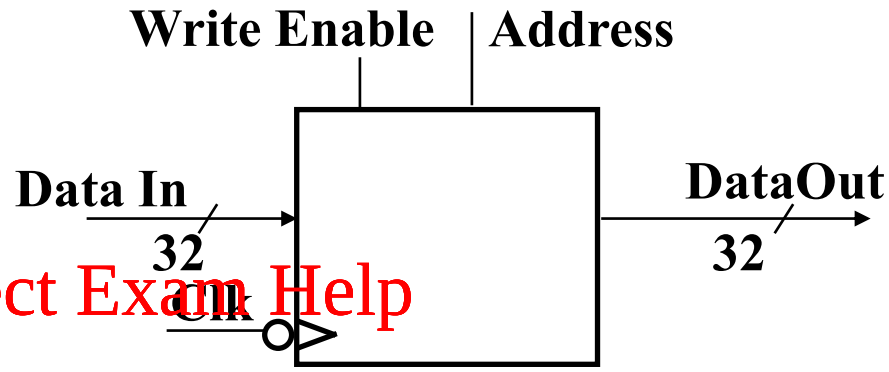
***A-B == 0 ?***

- Textbook also adds AND, SLT (1 if A < B, else 0)
- ALU follows Chapter 3 / Appendix C.5

Subtract, then use a giant nor… draw a truth table!

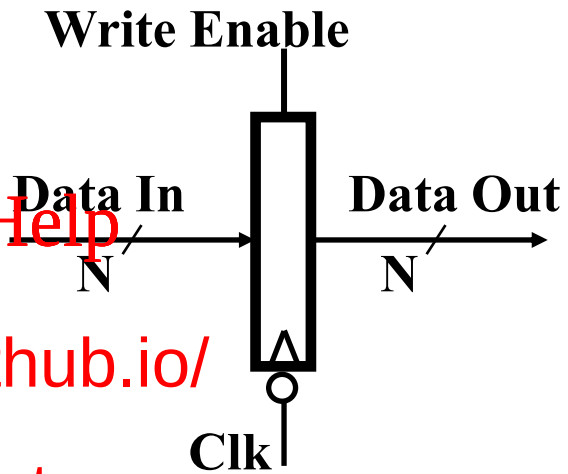
# Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is select
  - Address selects the wor
  - If **Write Enable = 1** then the address set d in memory to be written (it will be set to word on the **Data In** bus)
- Clock input (CLK)
  - The CLK input is a factor **ONLY** during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid ⇒ Data Out valid after "access time."



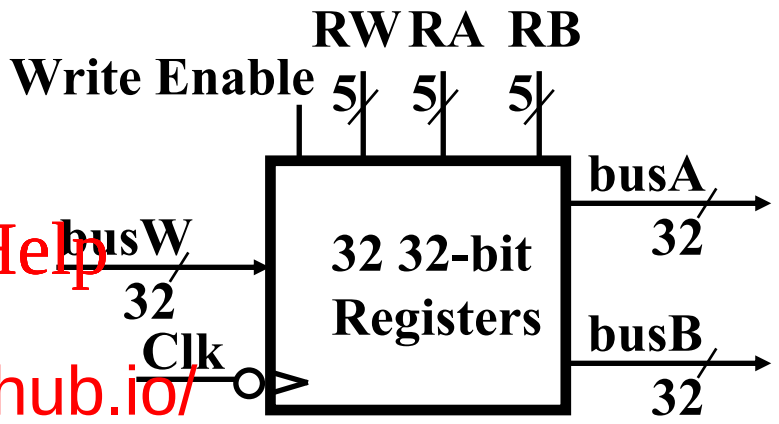
# Storage Element: Register (Building Block)

- Similar to D Flip Flop except
  - N-bit input and output
  - Write Enable input
- Write Enable:
  - When 0 Data Out will not change
  - When 1 Data Out will become Data I



# Storage Element: Register File

- Register File consists of 32 registers:
  - Two 32-bit output busses: busA, busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the r
  - RB (number) selects the register to put on b
  - RW (number) selects the register to be wri 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."



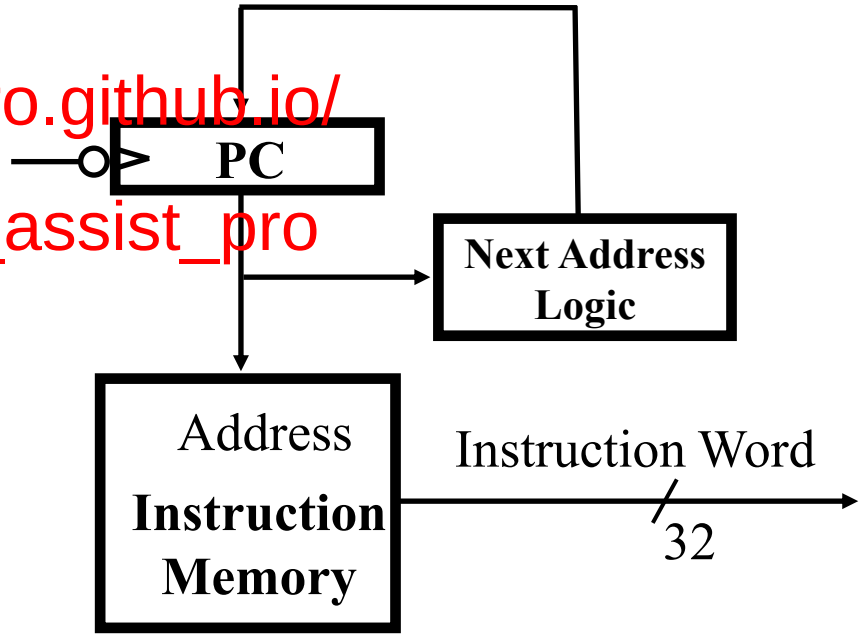
# Step 3: Assemble DataPath meeting requirements

- Register Transfer Requirements
  - ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and

# 3a: Overview of the Instruction Fetch Unit

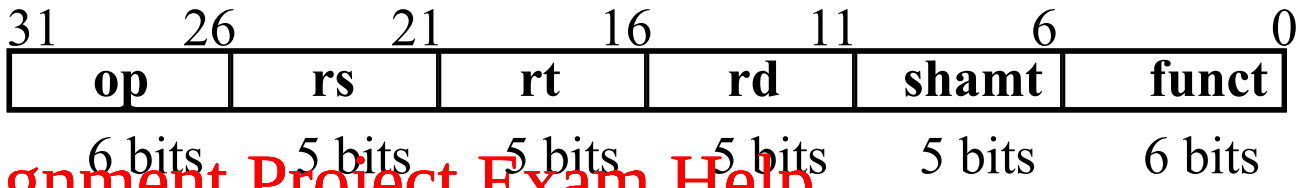
- Common register transfer language operations
  - Fetch the Instruction: mem[PC]
  - Update the program
    - Sequential Code:
      PC = PC + 4
    - Branch and Jump:
      PC = “something else”





# 3b: Add & Subtract

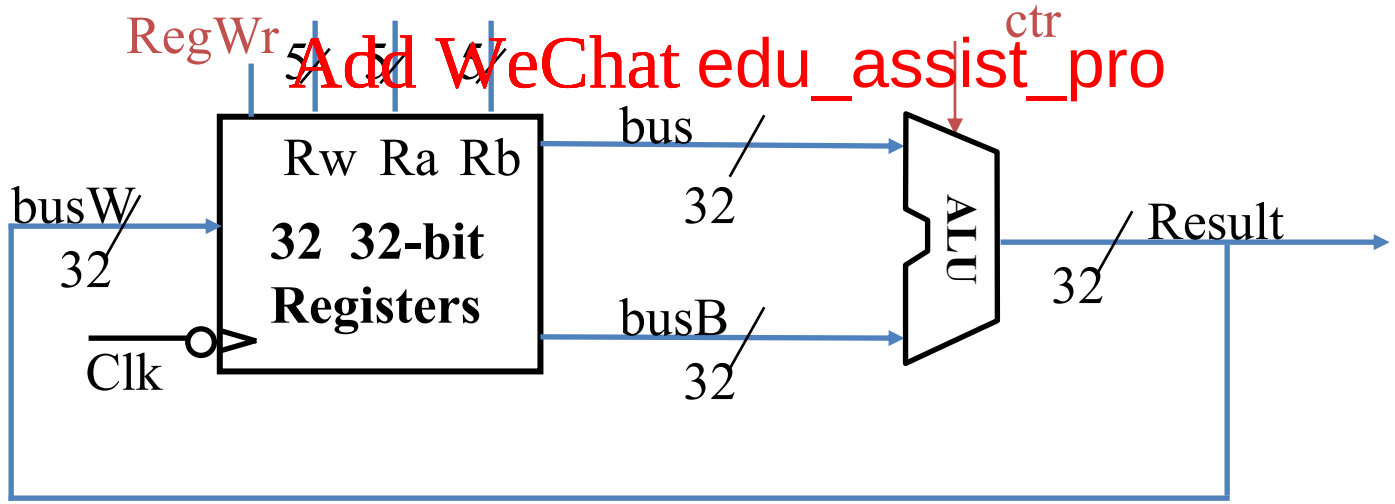
- R[rd] = R[rs] op R[rt]  Ex.: `addU rd,rs,rt`
  - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields






  - ALUctr and RegWr: co the instruction

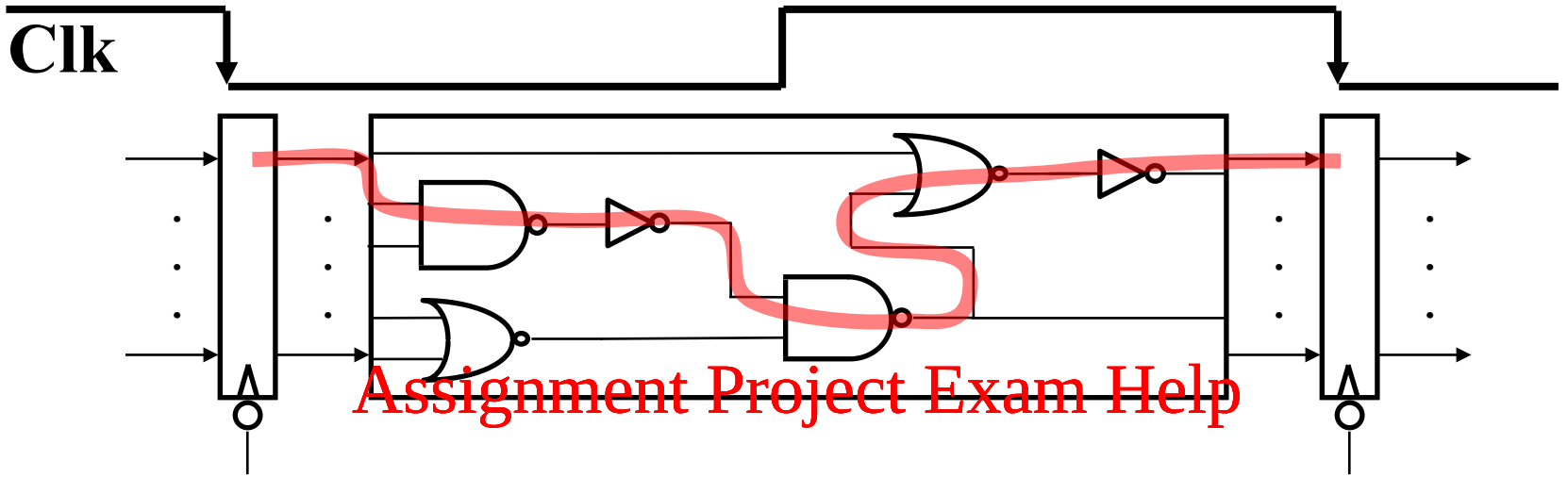







Already defined register file, ALU

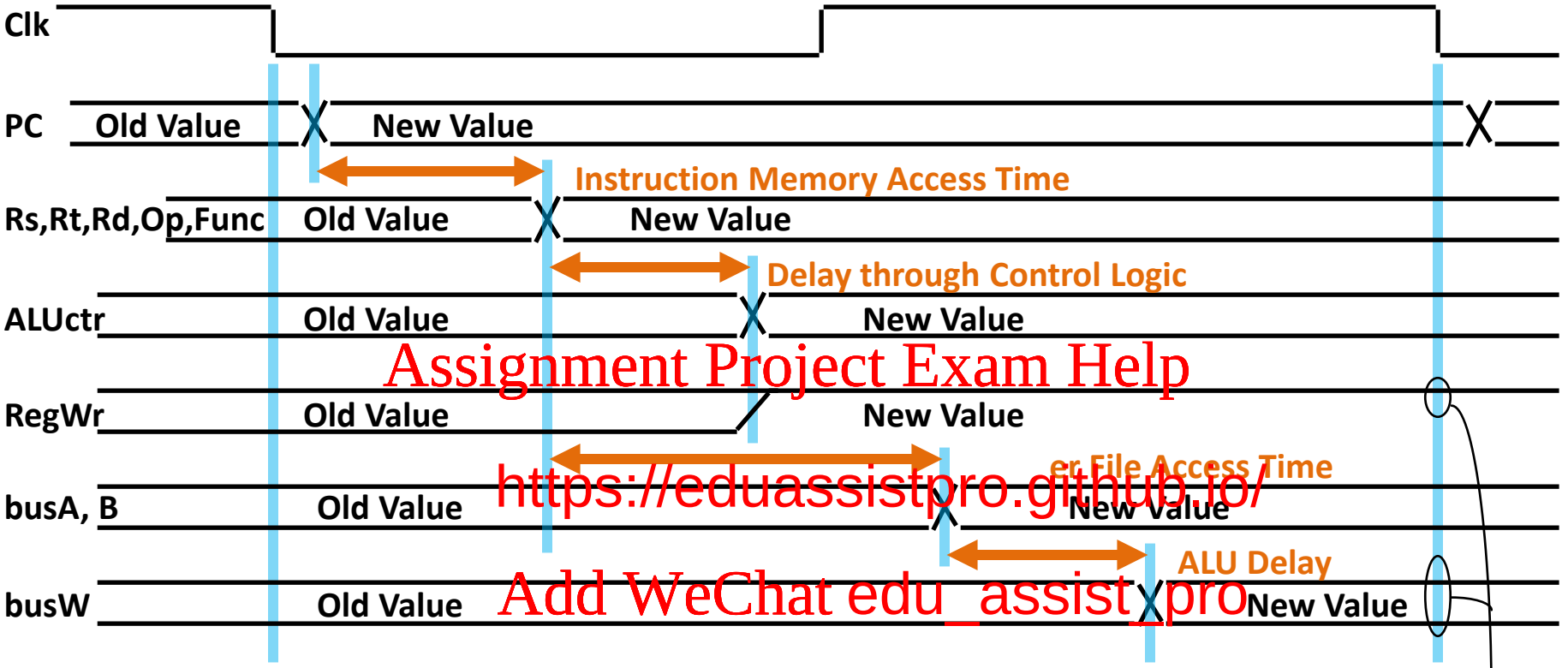
# Clocking Methodology



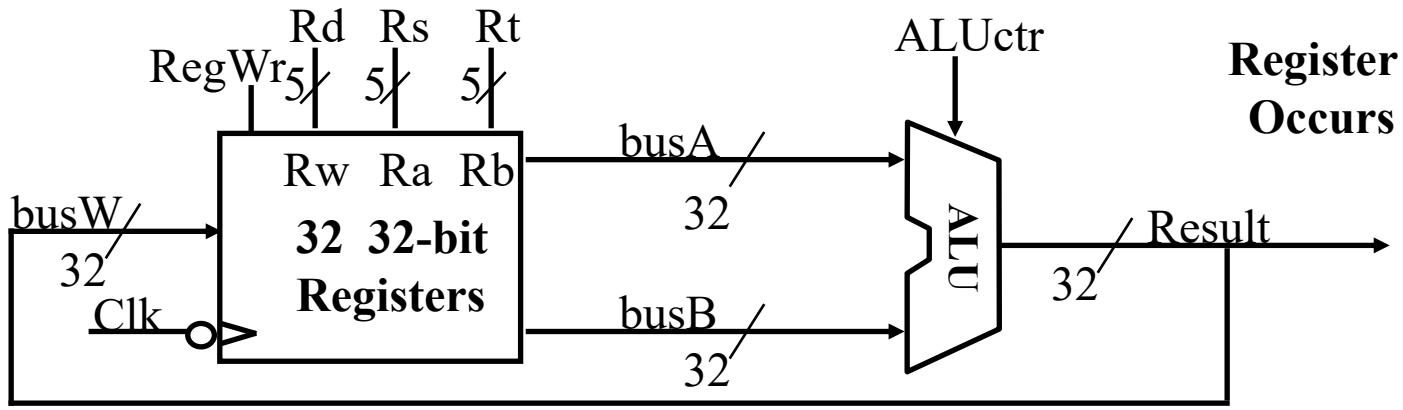



- Storage elements clocke
- Being physical devices, flip-flops (FF) an ional logic have some delay
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF, and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

# Register-Register Timing: One complete cycle

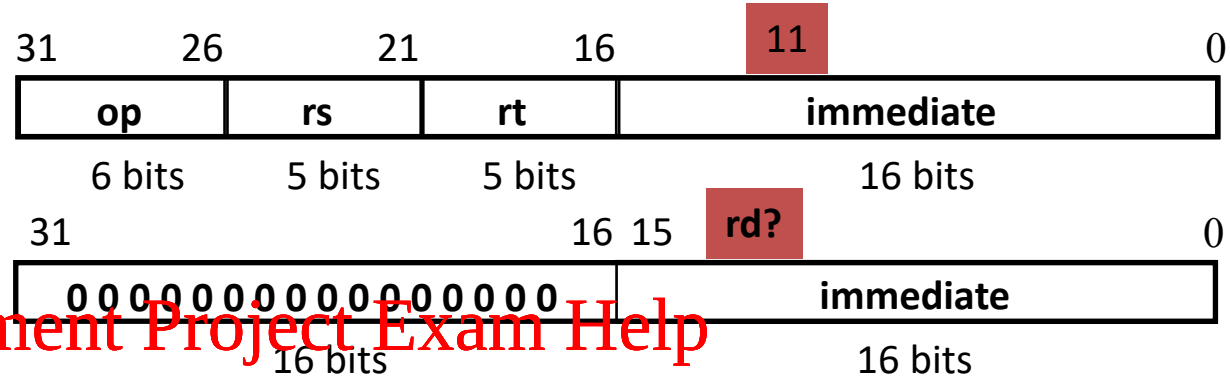


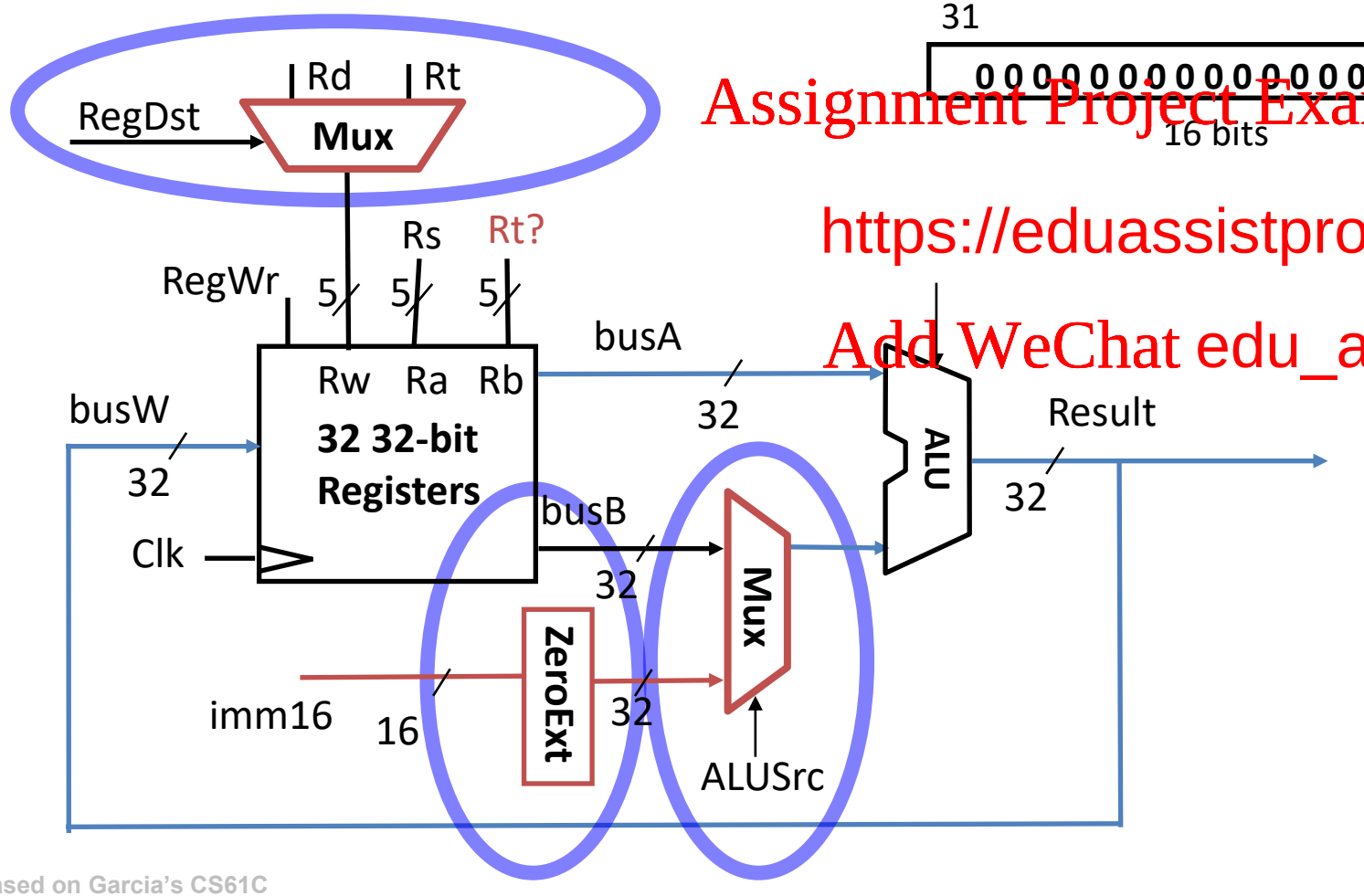


# 3c: Logical Operations with Immediate

- R[rt] = R[rs] op ZeroExt[imm16] ]



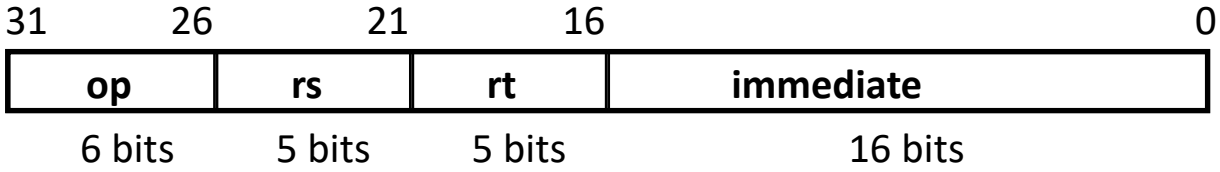



*register read??*
*hat happens?*
*Does it matter?*

- Need a 32-bit MUX to select the immediate
- Need to zero extend the 16 bit immediate for logical operation
- Need a 5-bit MUX to select the destination

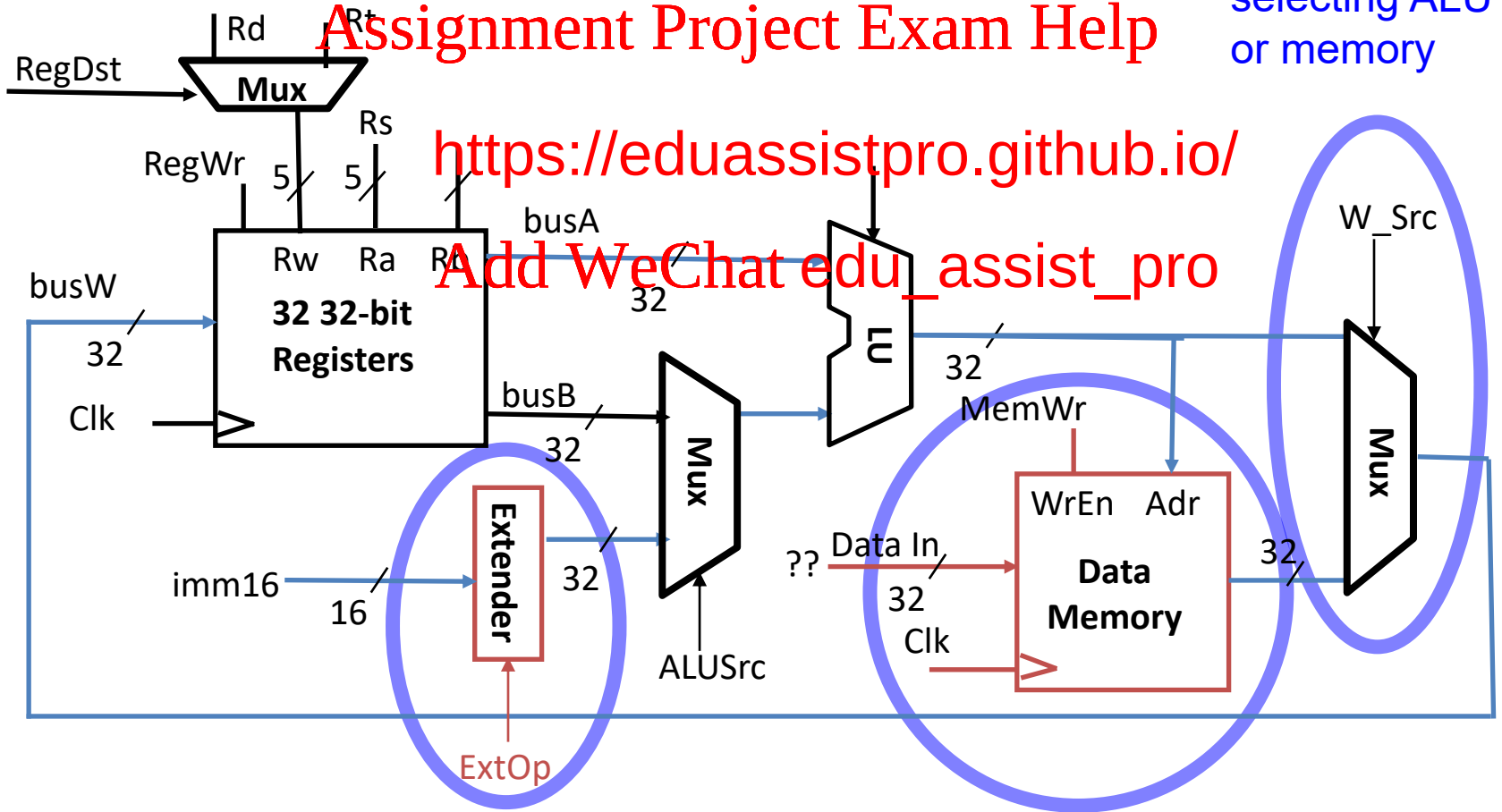
# 3d: Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



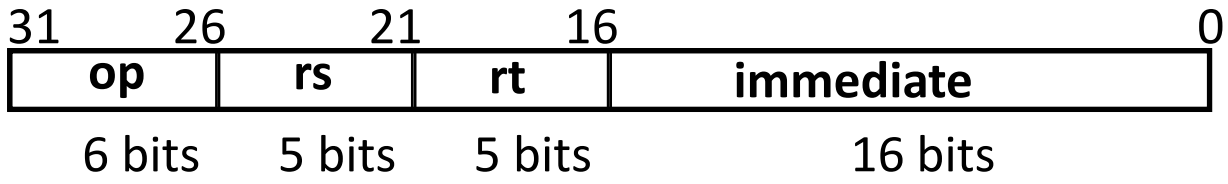

- Modify immediate extender to sign or zero extend based on ExtOp control signal
- Include memory in datapath
- Include mux for selecting ALU or memory



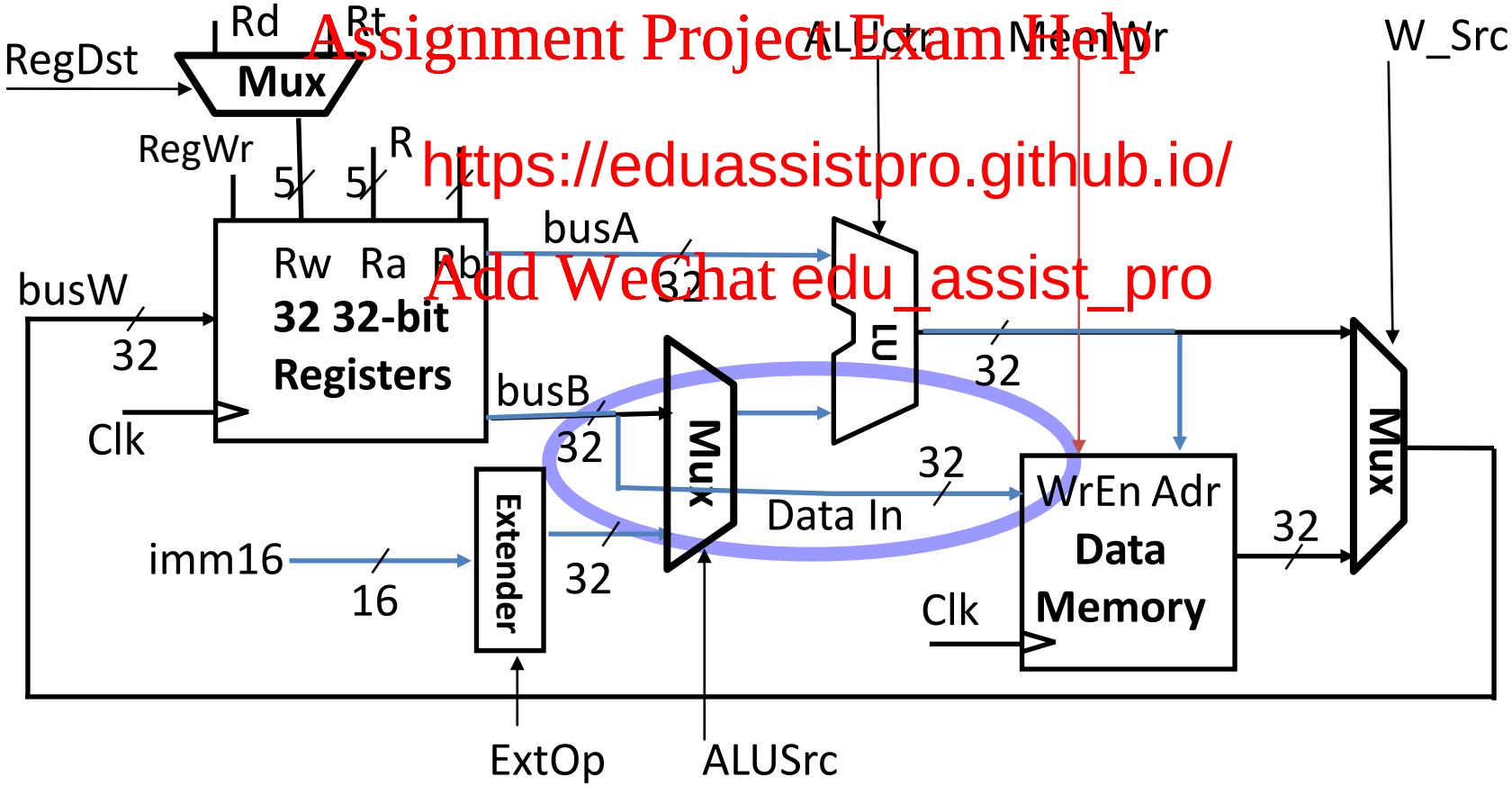


# 3e: Store Operations

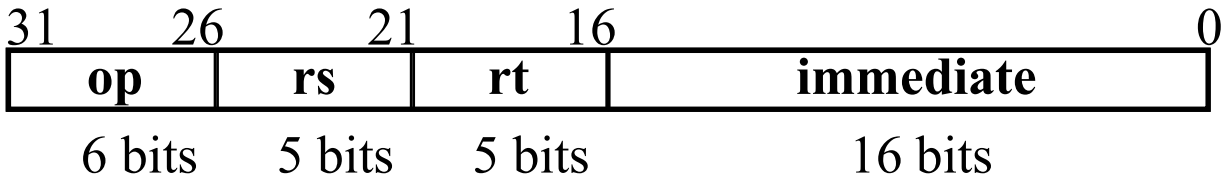
- Mem[ R[rs] + SignExt[imm16] ] = R[rt]
  Ex.: `sw rt, rs, imm16`




- For store to work we must connect busB to memory Data In
- Datapath now mostly complete!



# 3f: The Branch Instruction



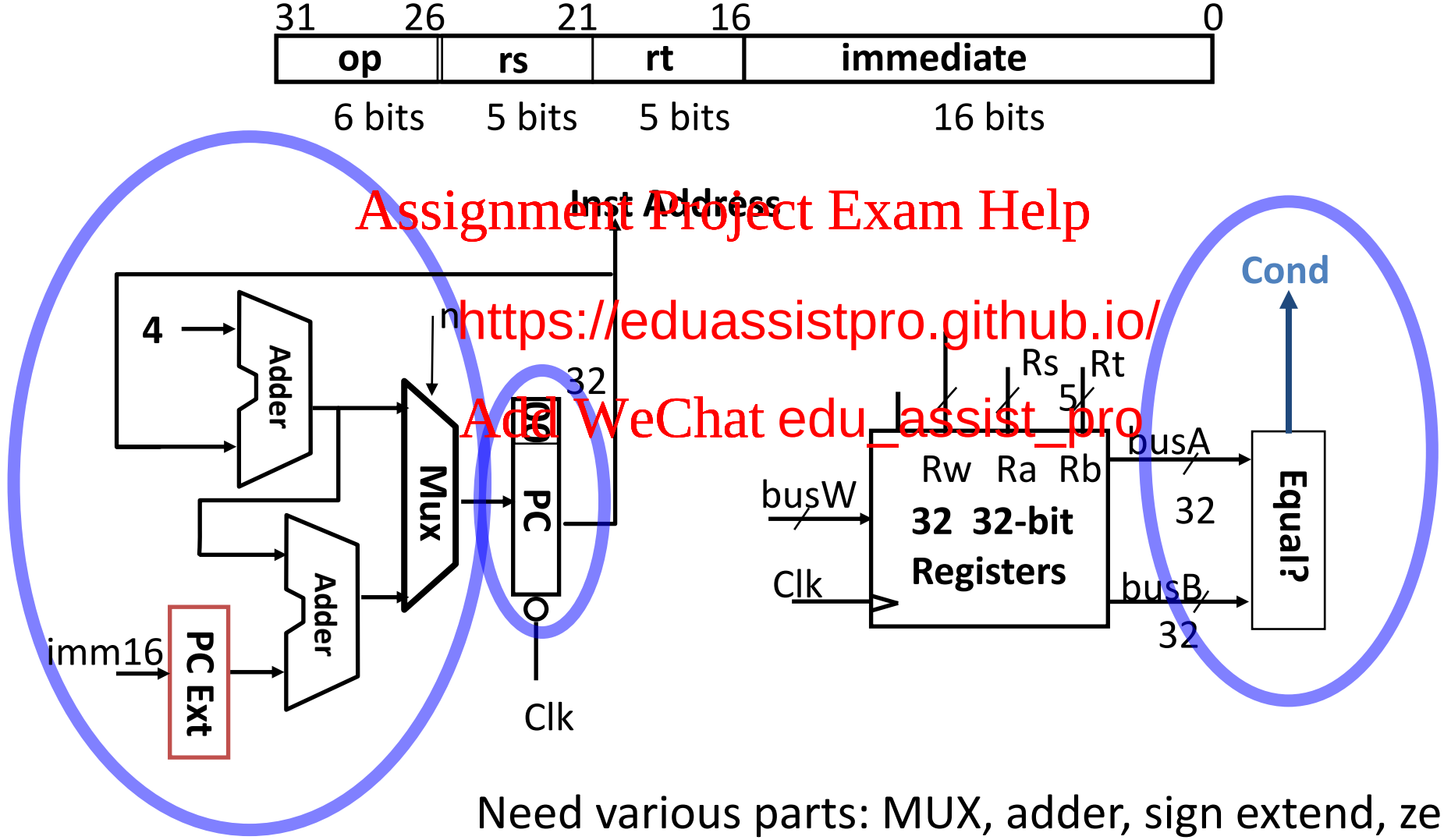

- `beq rs, rt, imm16`
  - mem[PC] Fetch the instruction from memory
  - Equal = R[rs] == R[rt] branch condition
  - if (Equal) Calculate the next instruction's address
    - PC = PC + 4 + ( SignExt(imm16) x 4 )

    else
    - PC = PC + 4

# Datapath for Branch Operations

- beq rs, rt, imm16

  Datapath generates condition (equal, e.g., zero on subtract)

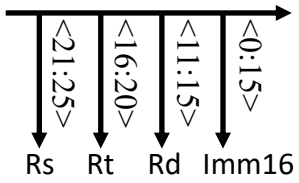




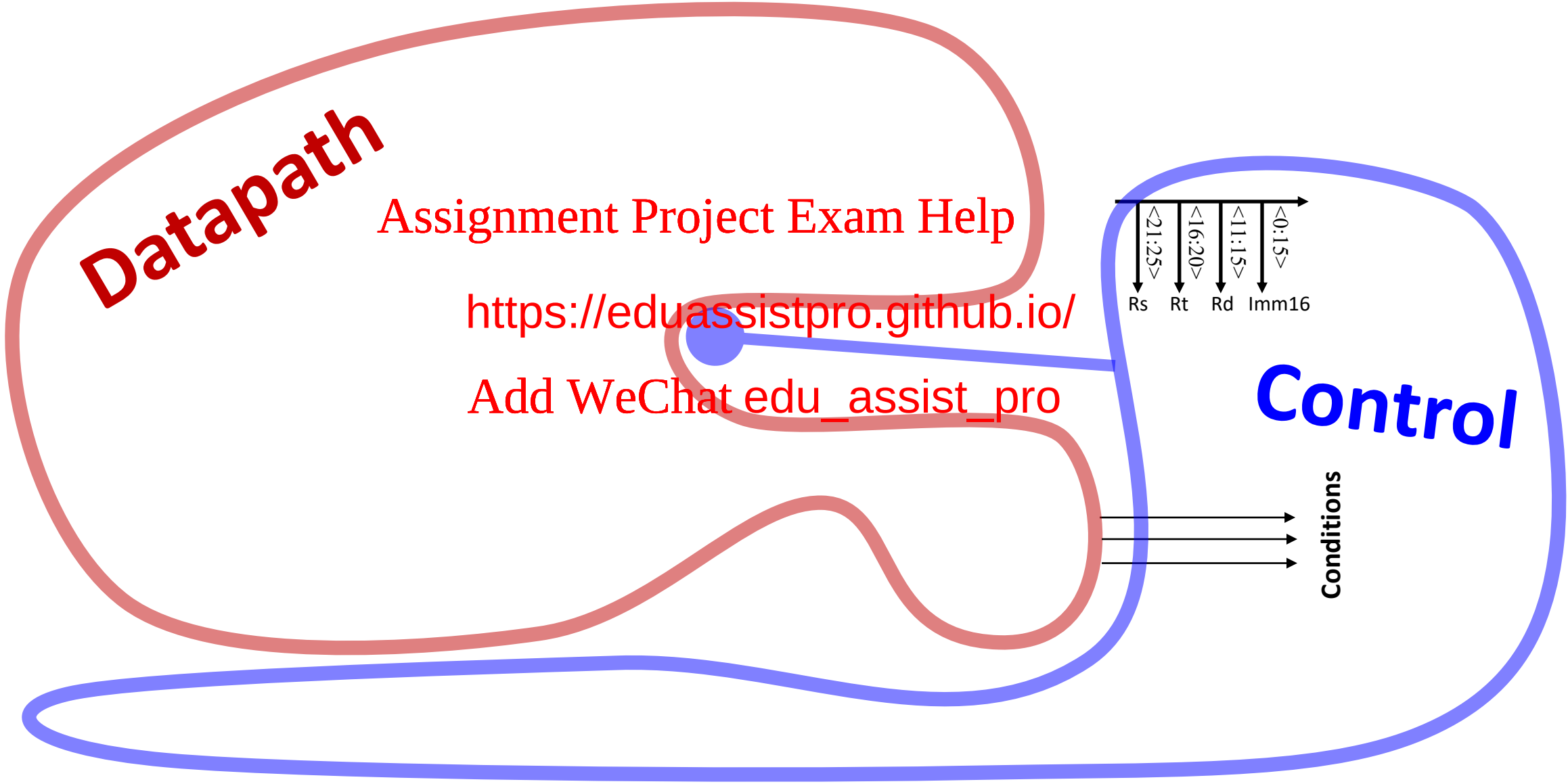

Need various parts: MUX, adder, sign extend, zero

# Putting it All Together: A Single Cycle Datapath!!





# An Abstract View of the Implementation



# Questions

A. If the destination reg is the same as the source reg, we could compute the incorrect value!

No, clocking preve

B. We're going to be ters and write a 3rd in 1 cycle

Yes

C. Datapath is hard, Control is easy

No, control is hard

# Questions

A. Our ALU is a synchronous device

No, it is a combinatorial circuit



B. We should use the te PC=PC+4



No, it is needed for other purpos



C. The ALU is inactive for memory reads or writes.

No, the ALU is used to compute the offset

# Questions

A. SW can peek at HW (past ISA abstraction boundary) for optimizations

Probably

B. SW can depend on pa tation of ISA

Probably not

C. Timing diagrams serve as a critical debugging tool in design of circuits

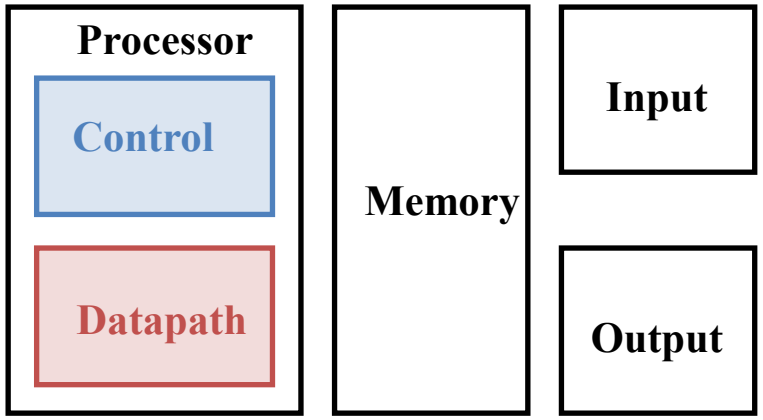
Yes



# Summary: Single cycle datapath

- 5 steps to design a processor
  1. Analyze instruction set ⇒ datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble dat ents
  4. Analyze impl n to determine setting of control points that effects th sfer
  5. Assemble the control logic
- Control is the hard part
- Next time!





# Review and More Information

- Textbook Chapter 4, Sections 4.1 to 4.3

# Extra Questions

1. (a'+b)• (a+b) = b
2. N-input gates can be thought of cascaded 2-input gates. That is,
   (a Δ b Δ c Δ d
   where Δ is o AND
3. You can use NOR(s) with cl g_b simulate AND, OR, & NOT
4. During read operation, the register file behaves as a combinational logic block



# Extra Questions, True or False

1. Truth table for mux with 4-bits of signals has $2^4$ rows
2. We could [illegible] ters to make 1 srl
3. If 1-bit adder delay is T, it adder delay would also be T
4. Virtual memory would be impossible without a TLB