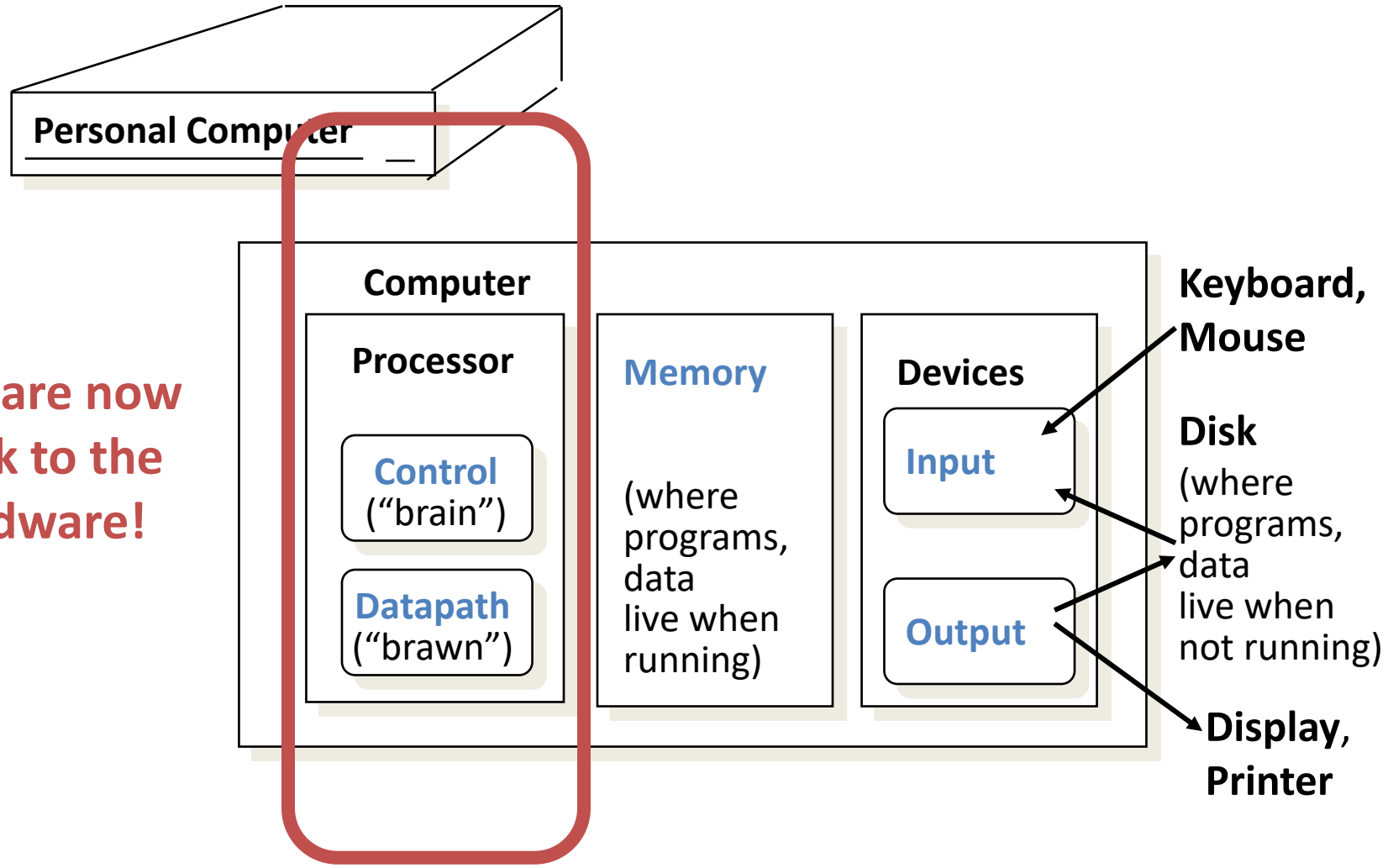
Single Cycle CPU Datapath

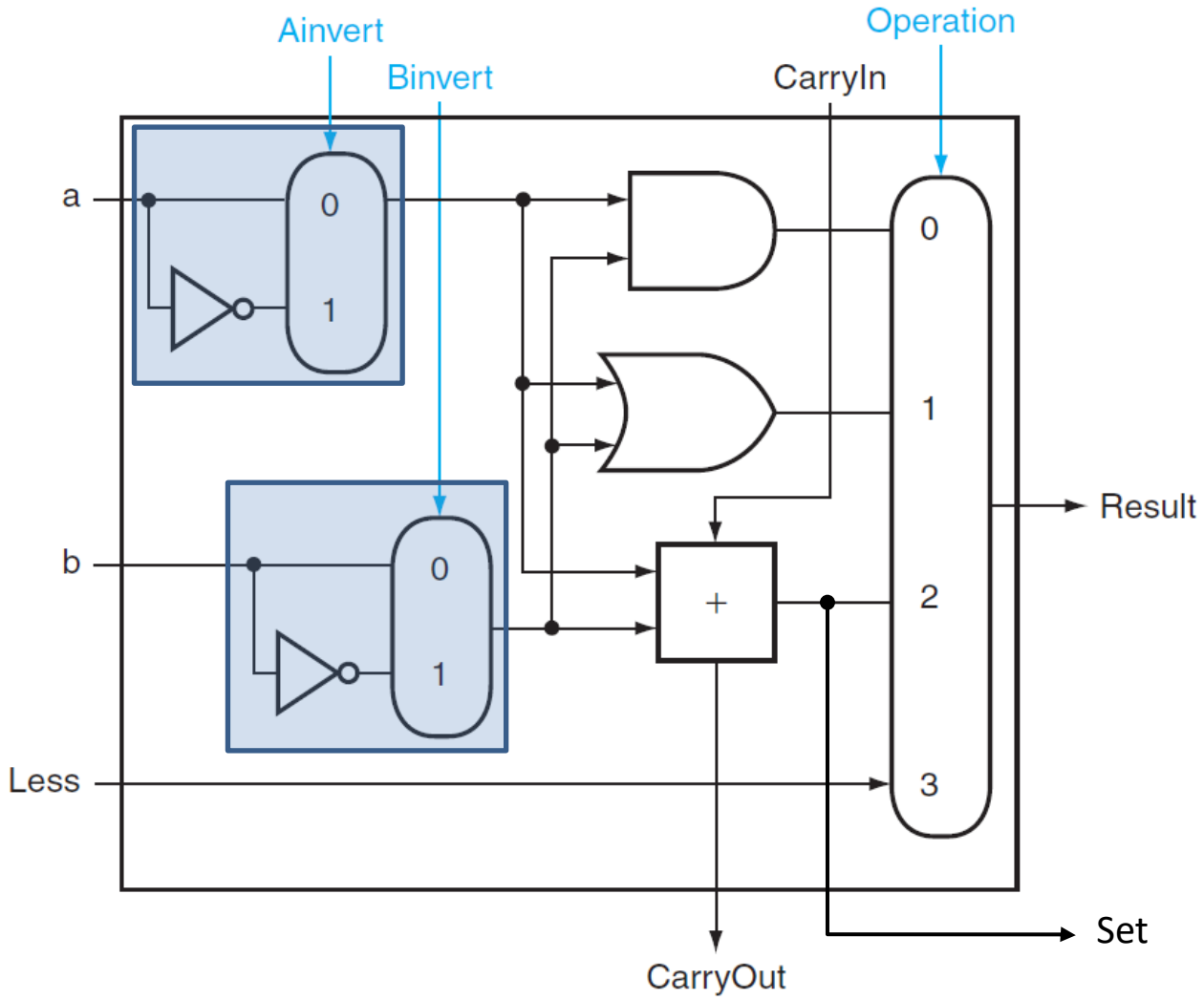
COMP273

Review: 5 parts of any Computer



We are now back to the hardware!

Review, 1 bit ALU



For SLT

Instruction	Ainvert	Binvert	CarryIn	Operation
ADD	0	0	0	2
SUB	0	1	1	2
AND	0	0	x	0
OR	0	0	x	1
NOR	1	1	x	0
SLT	0	1	1	3

Based on Garcia's CS61C

Outline

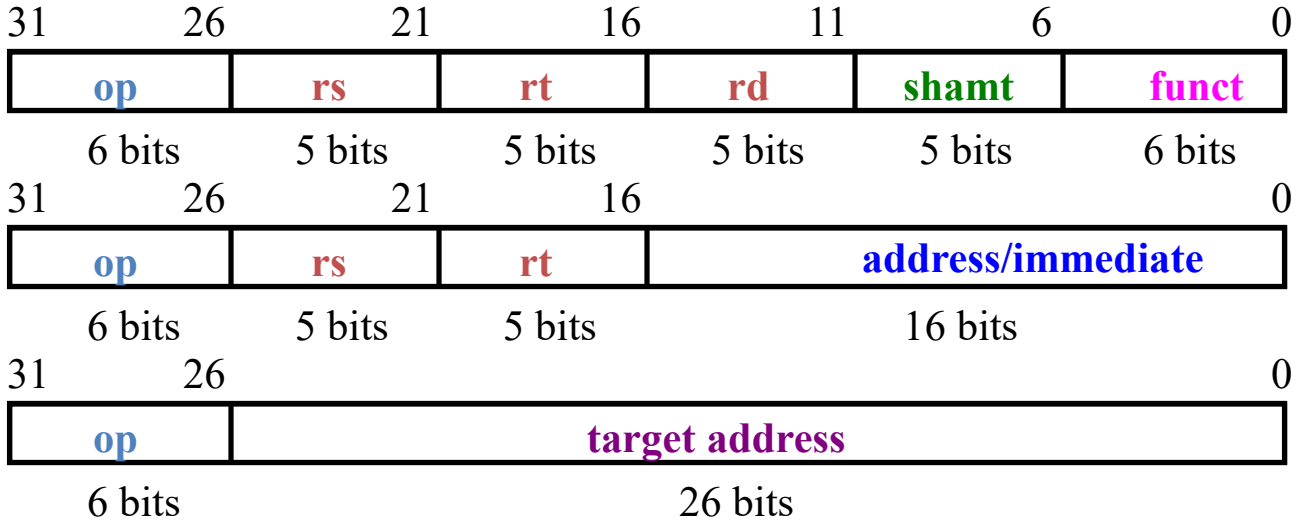
- Design a processor: step-by-step
- Requirements of the Instruction Set
- Hardware components that match the instruction set requirements

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 each register transfer
2. Select set of datapath components and establish 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

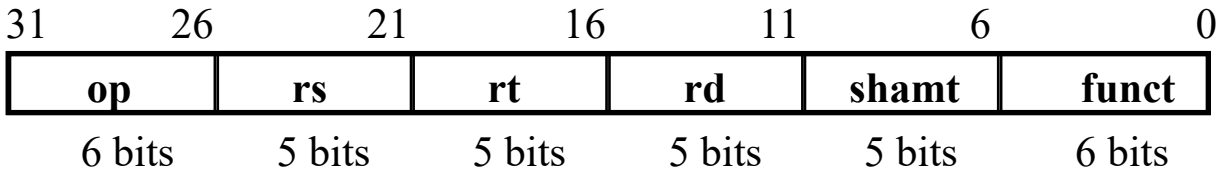
- All MIPS instructions are 32 bits long, 3 formats:
  - R-type
  - I-type
  - J-type

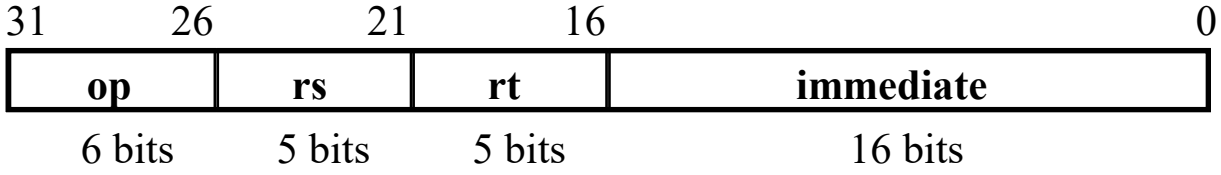


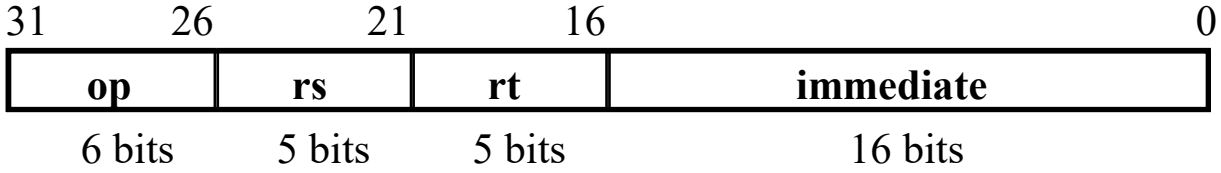
- The different fields are:
  - op: operation ("opcode") of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: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

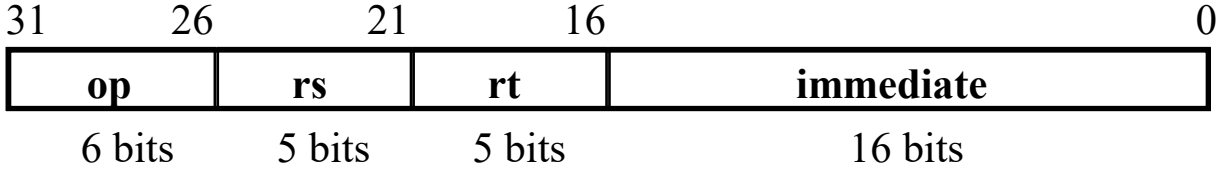
Step 1a: The MIPS-lite Subset for today

- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`
- OR Immediate:
  - `ori rt,rs,imm16`
- 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

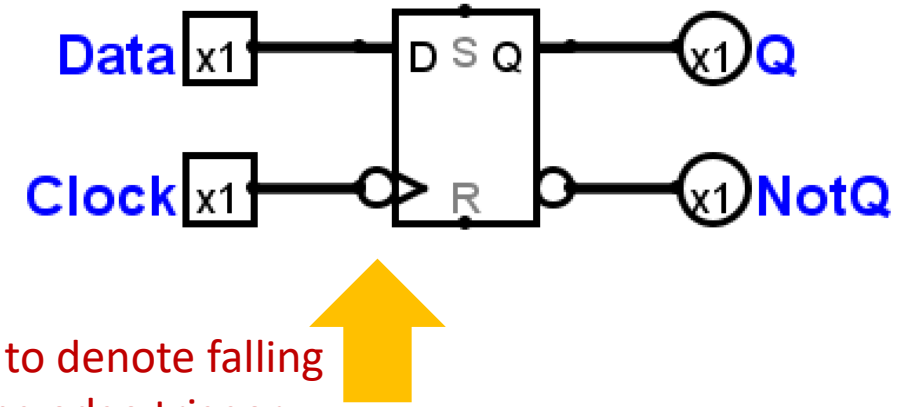
Instruction	Register Transfers
ADDU	R[rd] = R[rs] + R[rt]; PC = PC + 4
SUBU	R[rd] = R[rs] – R[rt]; PC = PC + 4
ORI	R[rt] = R[rs] | zero_ext(Imm16); PC = PC + 4
LW	R[rt] = MEM[R[rs] + sign_ext(Imm16)]; PC = PC + 4
SW	MEM[R[rs] + sign_ext(Imm16)] = R[rt]; PC = PC + 4
BEQ	if (R[rs] == R[rt]) then PC = PC + 4 + (sign_ext(Imm16) || 00 (i.e., *4)) else PC = PC + 4

Step 1: Requirements of the Instruction Set

- Memory (MEM)
  - instructions & data
- Registers (R: 32 x 32 bits)
  - 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

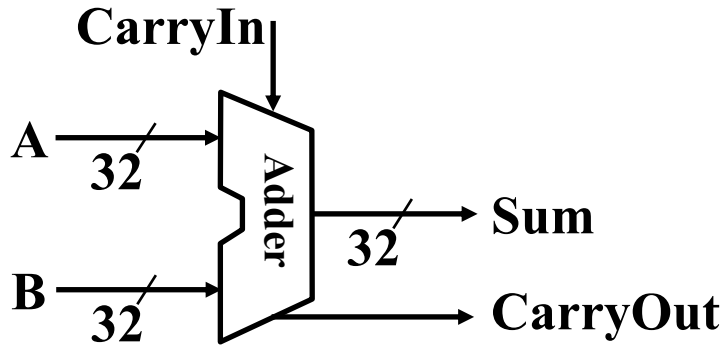
- Combinational Elements
- Storage Elements
  - Clocking methodology
    - We will use falling edge triggered element in these examples, thus you will see a small circle in front of the clock pin.



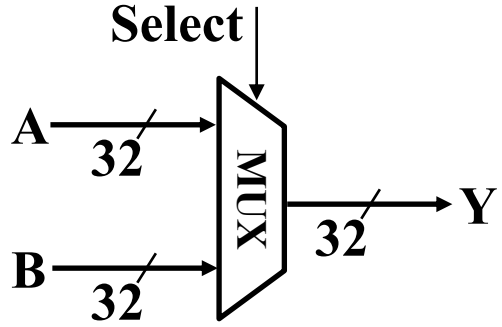
Logisim puts a negation circle in front of the clock to denote falling edge trigger. The circle is absent if you have a rising edge trigger.

Combinational Logic Elements (Building Blocks)

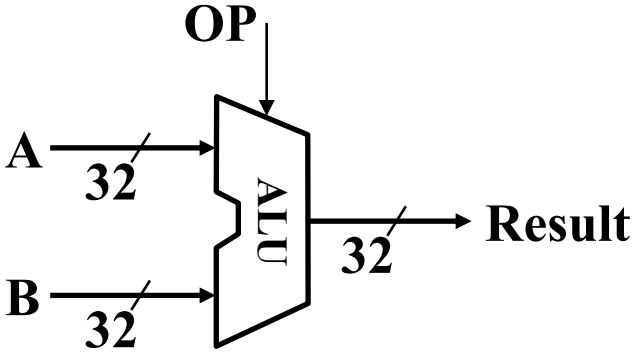
- Adder



- 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] | zero_ext(Imm16)...
BEQ   if ( R[rs] == R[rt] )...
```

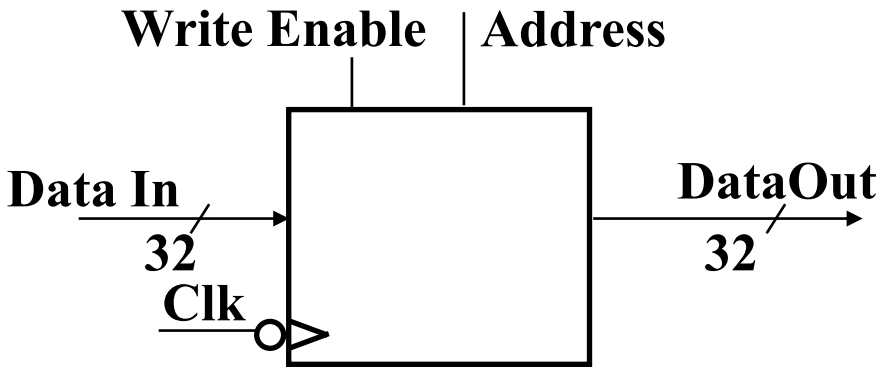
- Test to see if output == 0 for any ALU operation lets us implement the == equality test. ***How?***

A-B == 0 ?

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

- Textbook also adds AND, SLT (1 if A < B, else 0)
- ALU follows Chapter 3 (Arithmetic for Computers) and Appendix C.5 (constructing a Basic Arithmetic Logic Unit

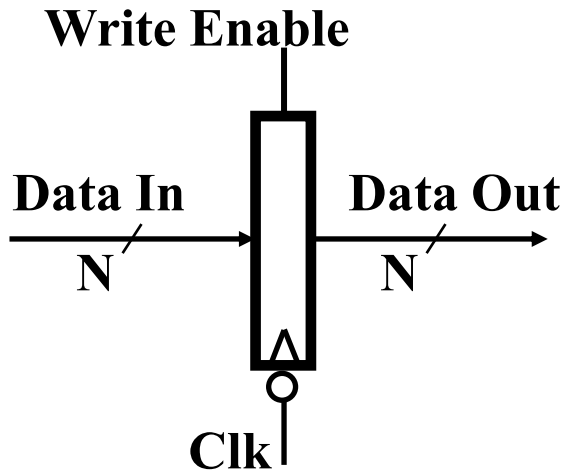
Storage Element: Idealized Memory



- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by address
  - Address selects the word to put on Data Out
  - If **Write Enable = 1** then the address selects the word 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 $\Rightarrow$ Data Out valid after "access time."

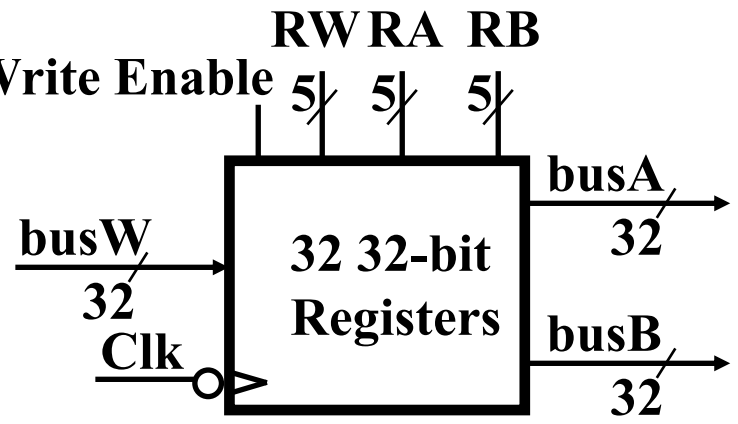
Storage Element: Register (Building Block)

- 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 In



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 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."

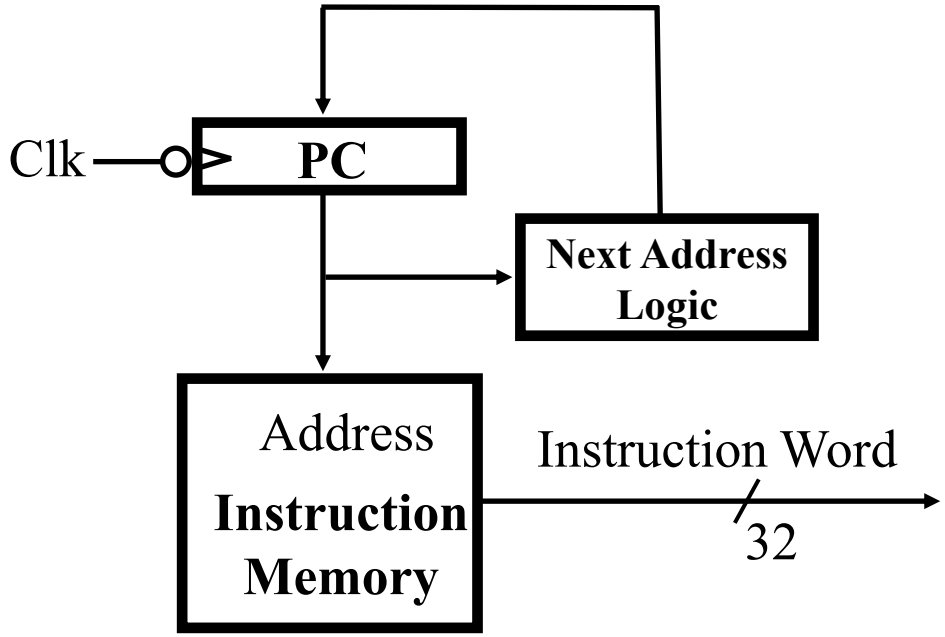


Step 3: Assemble DataPath meeting requirements

- Register Transfer Requirements
  $\Rightarrow$ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation

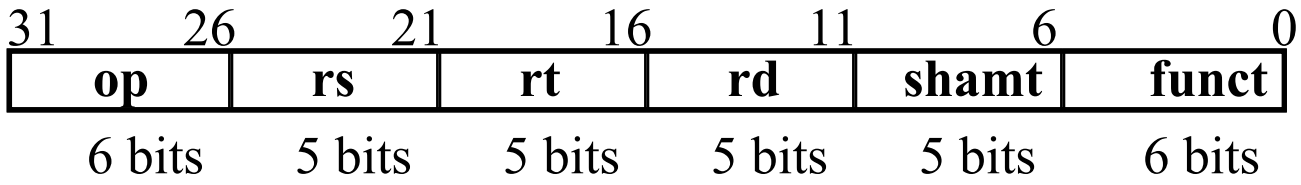
3a: Overview of the Instruction Fetch Unit

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

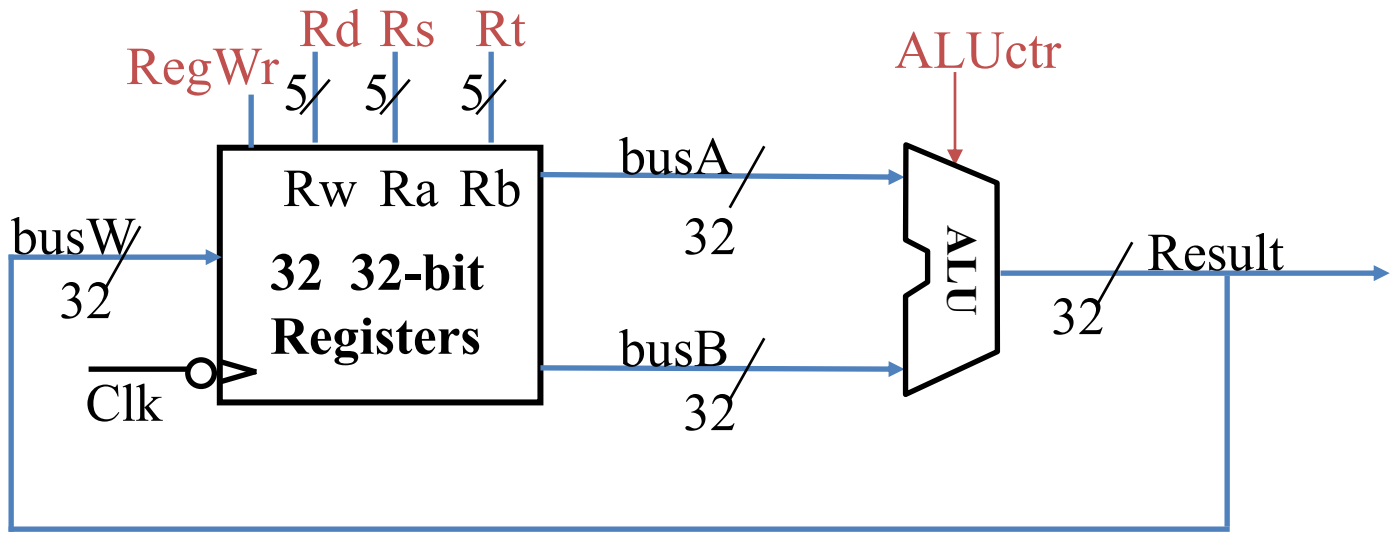


3b: Add & Subtract

- 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

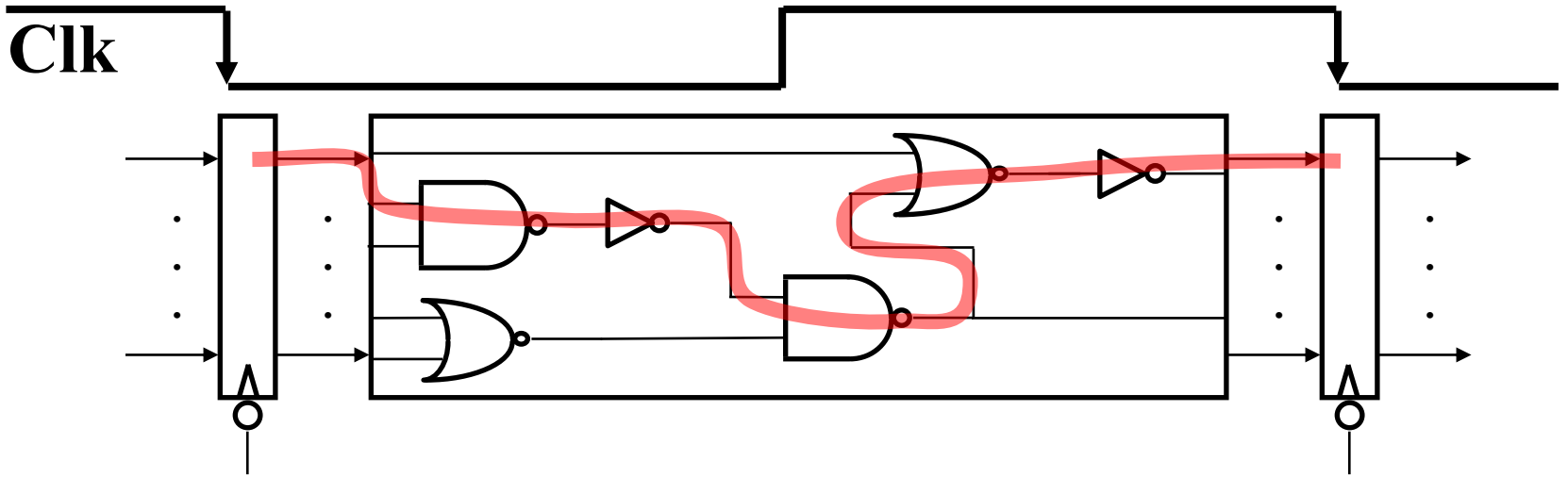


  - ALUctr and RegWr: control logic after decoding the instruction



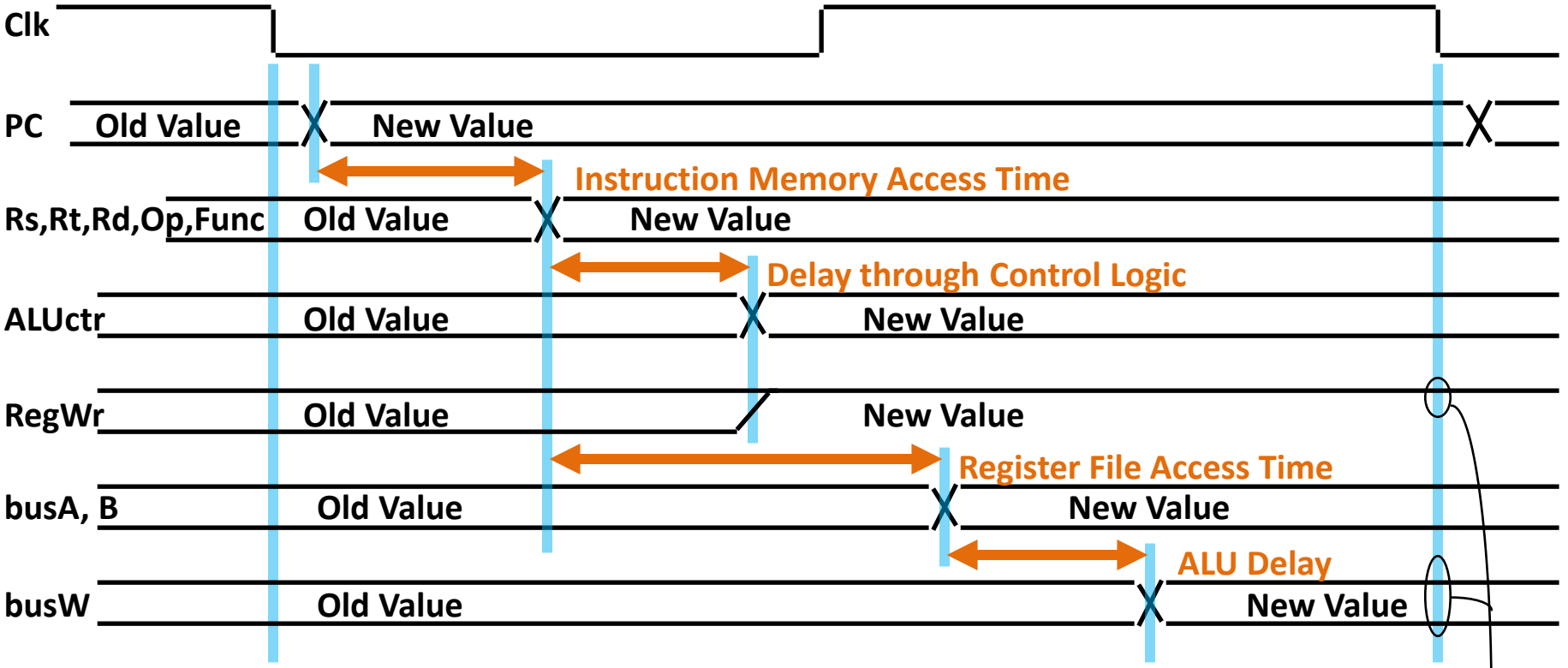
Already defined register file, ALU

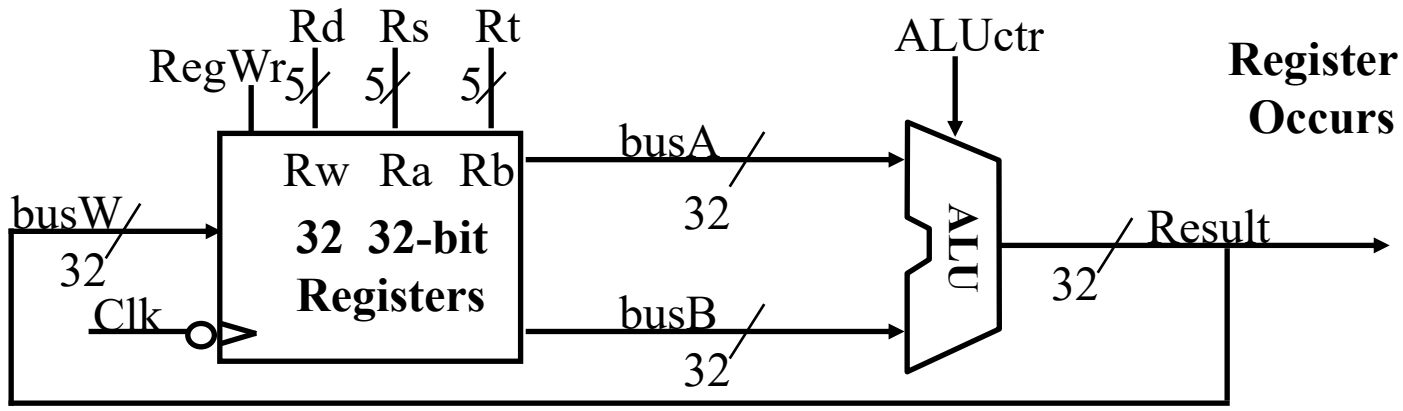
Clocking Methodology



- Storage elements clocked by same edge
- Being physical devices, flip-flops (FF) and combinational 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

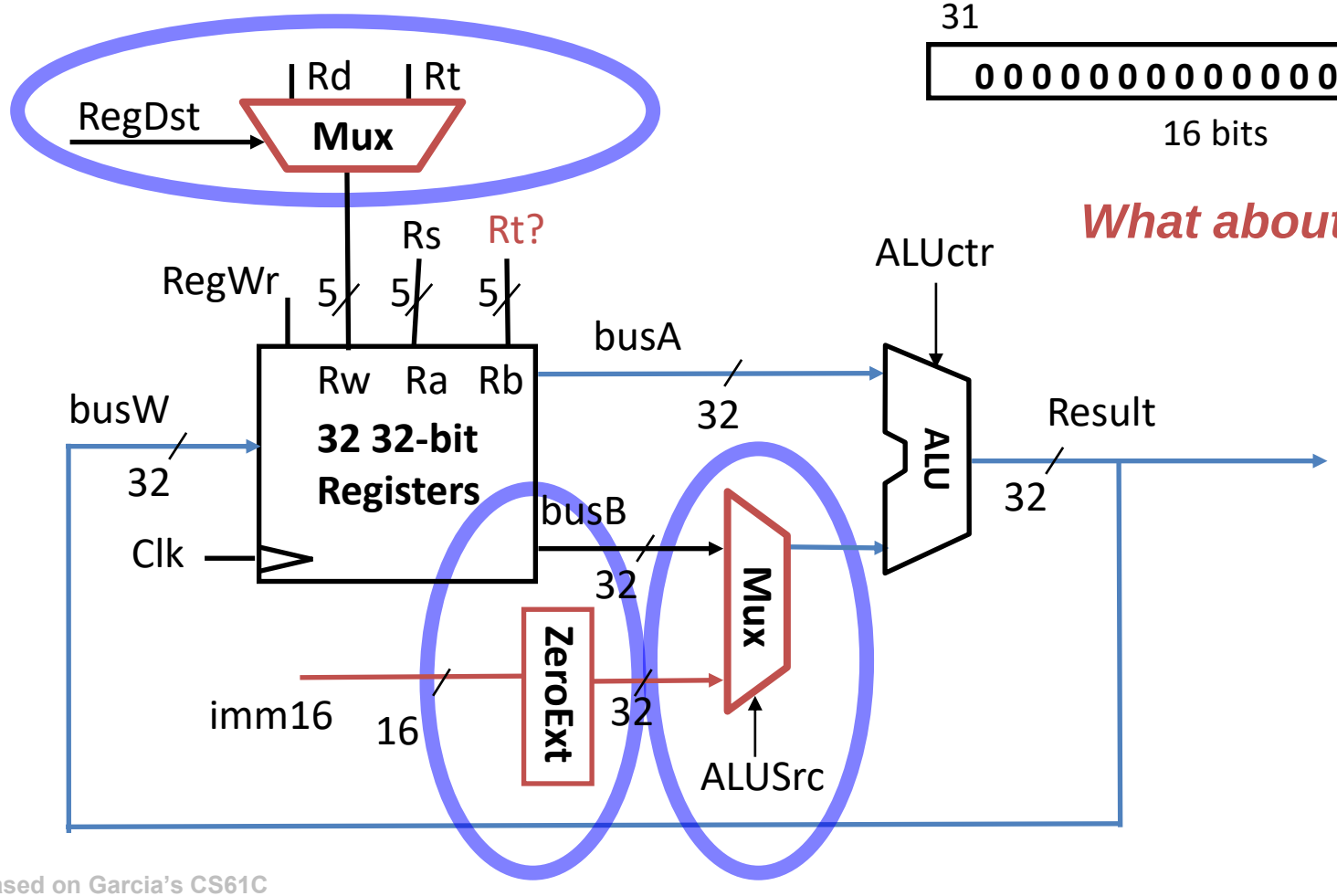
Register-Register Timing: One complete cycle

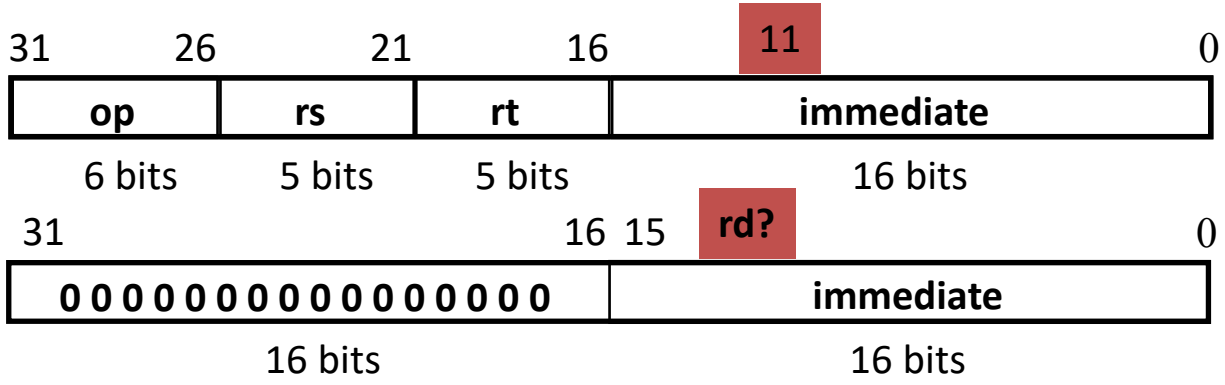




3c: Logical Operations with Immediate

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



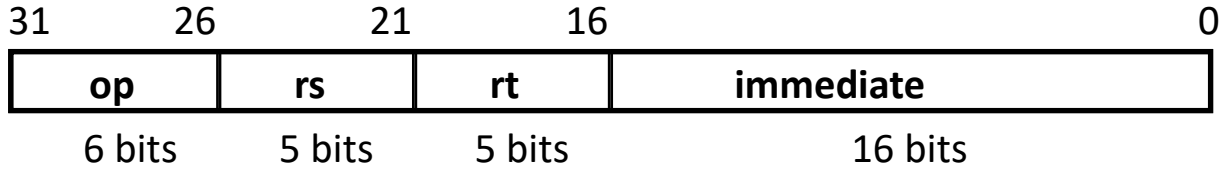


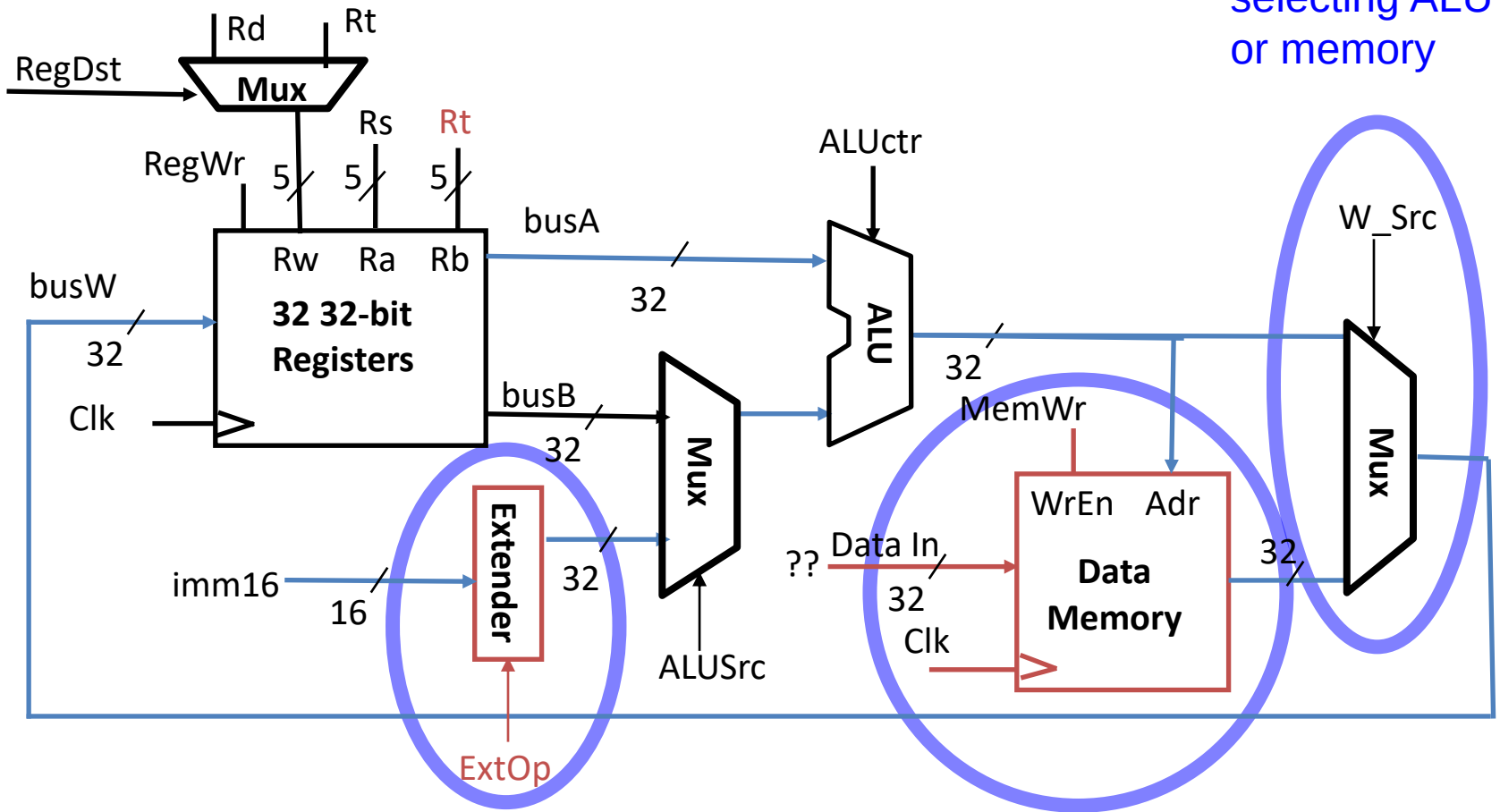
What about Rt register read??
What happens?
Does it matter?

- 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`

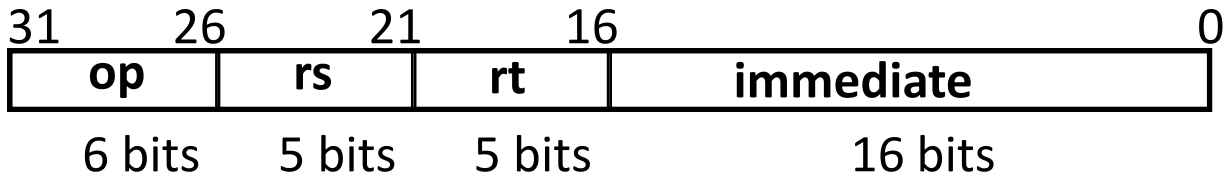


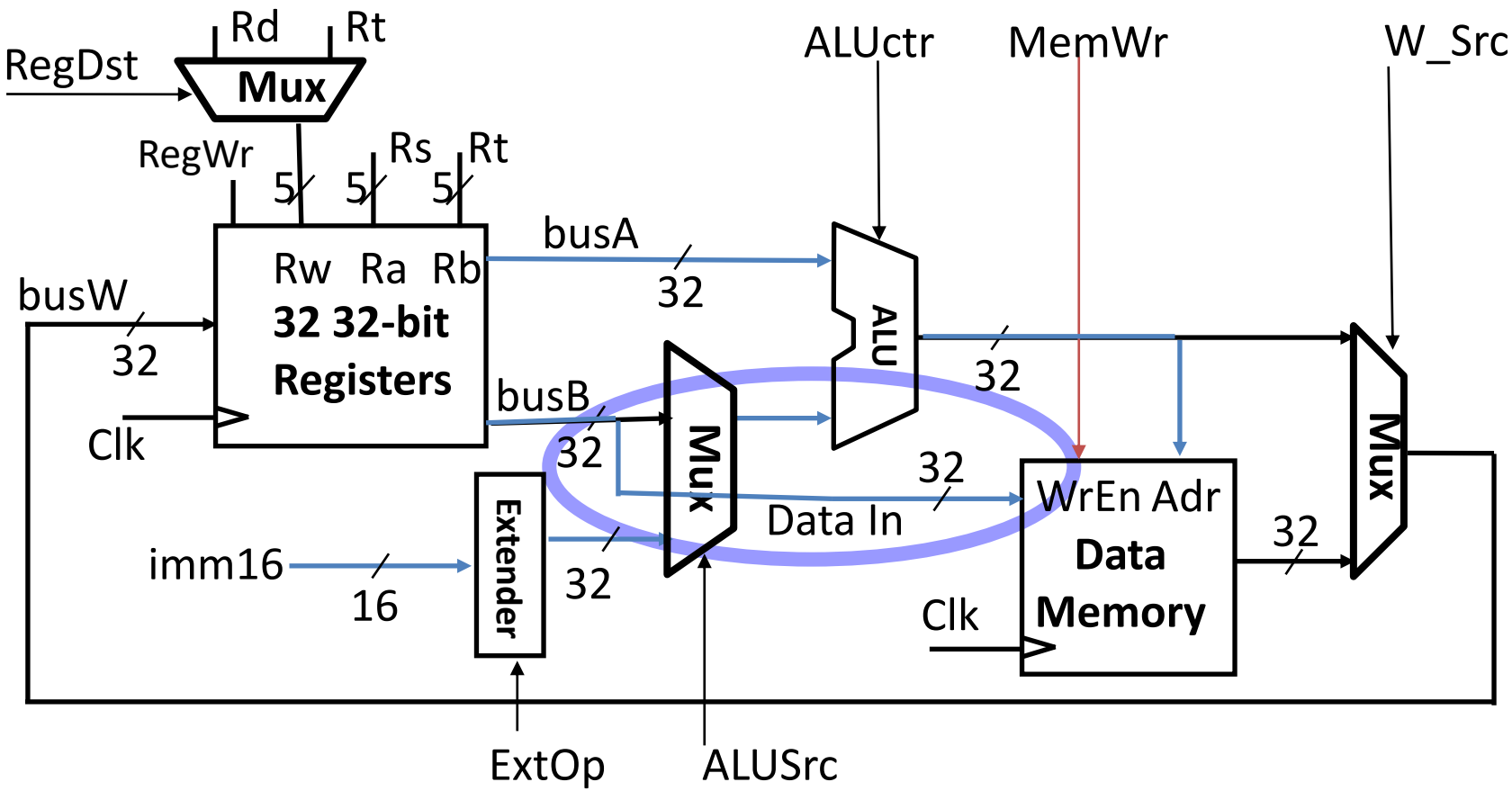


- Modify immediate extender to sign or zero extend based on ExtOp control signal
- Include memory in datapath
- Include mux for selecting ALU or memory

3e: Store Operations

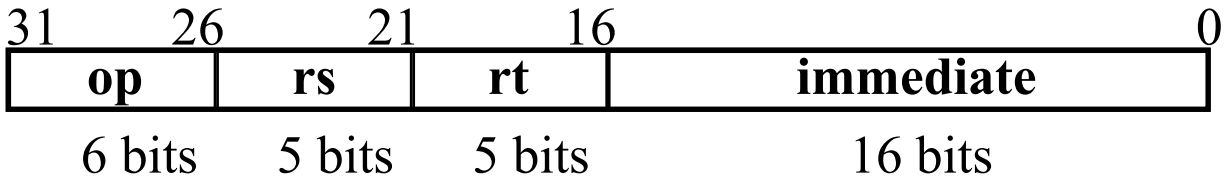
- Mem[R[rs] + SignExt[imm16]] = R[rt]
  Ex.: `sw rt, rs, imm16`





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

3f: The Branch Instruction



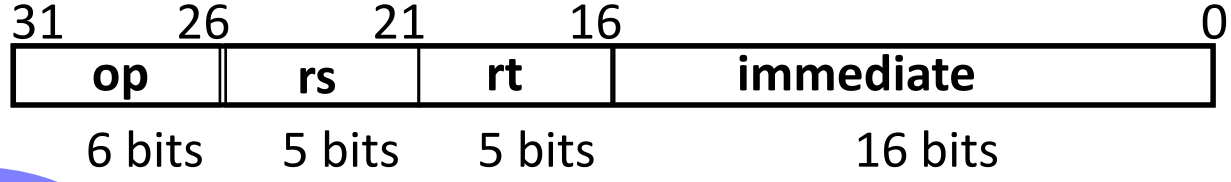
- `beq rs, rt, imm16`
  - mem[PC] Fetch the instruction from memory
  - Equal = R[rs] == R[rt] Calculate branch condition
  - if (Equal) Calculate the next instruction's address
    - PC = PC + 4 + (SignExt(imm16) x 4)

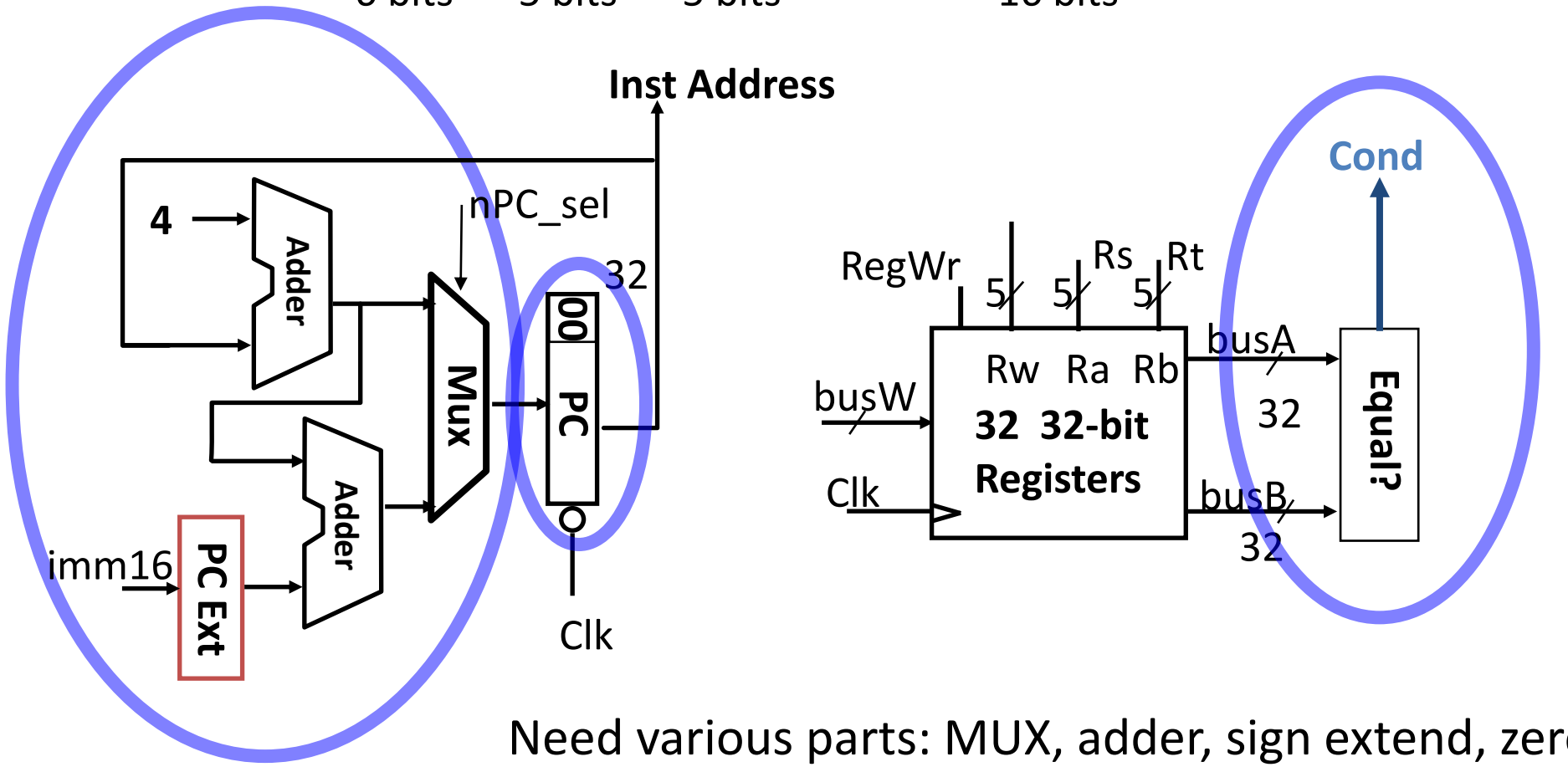
  else
    - PC = PC + 4

Datapath for Branch Operations

- beq rs, rt, imm16

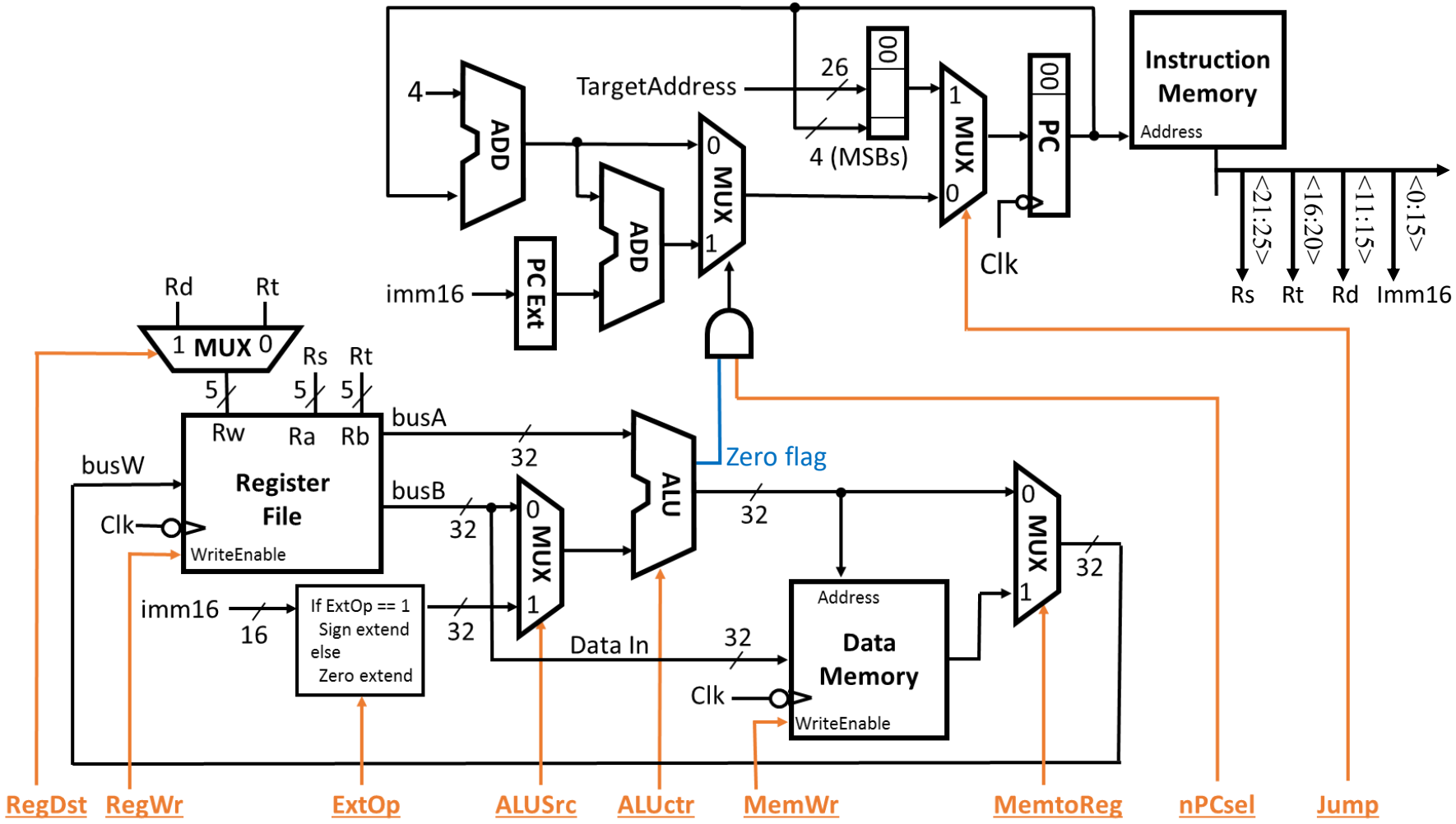
  Datapath generates condition (equal, e.g., zero on subtract)



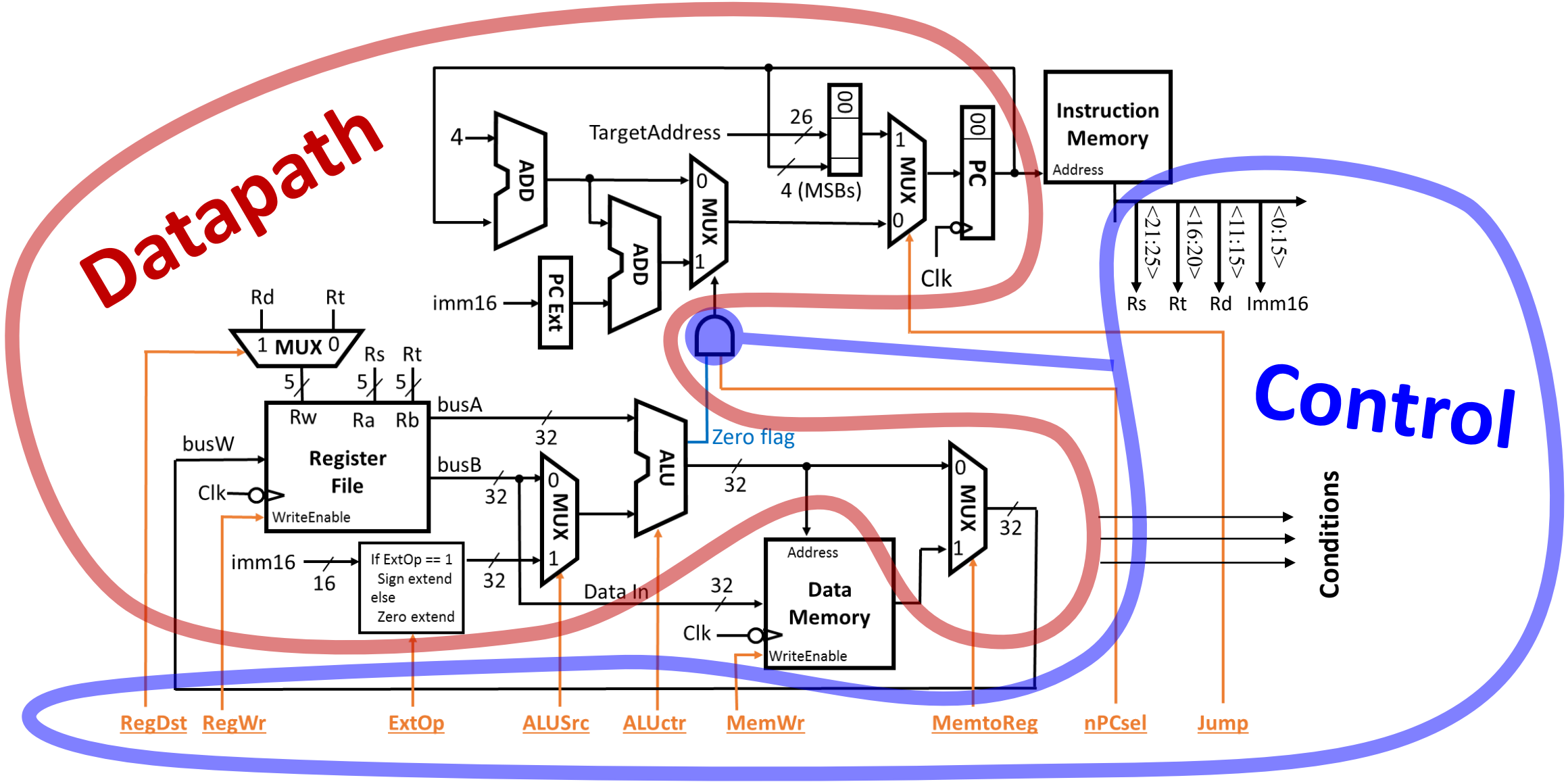


Need various parts: MUX, adder, sign extend, zero

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 prevents this

B. We're going to be able to read 2 registers 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 main ALU to compute PC=PC+4

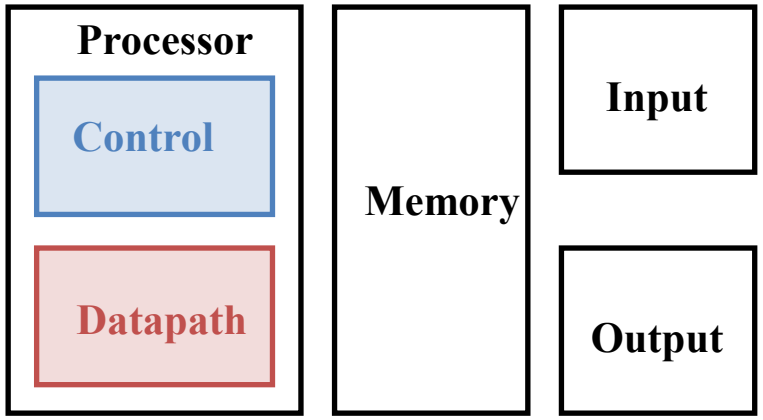
   No, it is needed for other purposes

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

   No, the ALU is used to compute the address by adding the offset to the base address.

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 datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
- Control is the hard part
- Next time!



Review and More Information

- Textbook Chapter 4 (The Processor)
  Sections 4.1 to 4.3
  - 4.1 Introduction
  - 4.2 Logic Design Conventions
  - 4.3 Building a Datapath