inst.eecs.berkeley.edu/~cs61c/su06

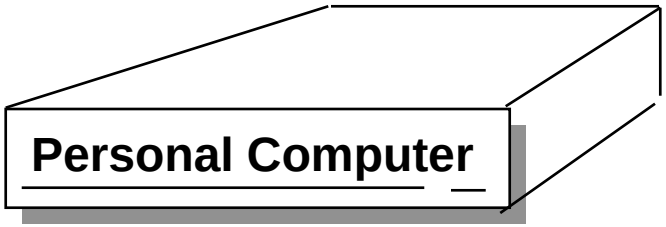
CS61C : Machine Structures

Lecture #16 – Datapath

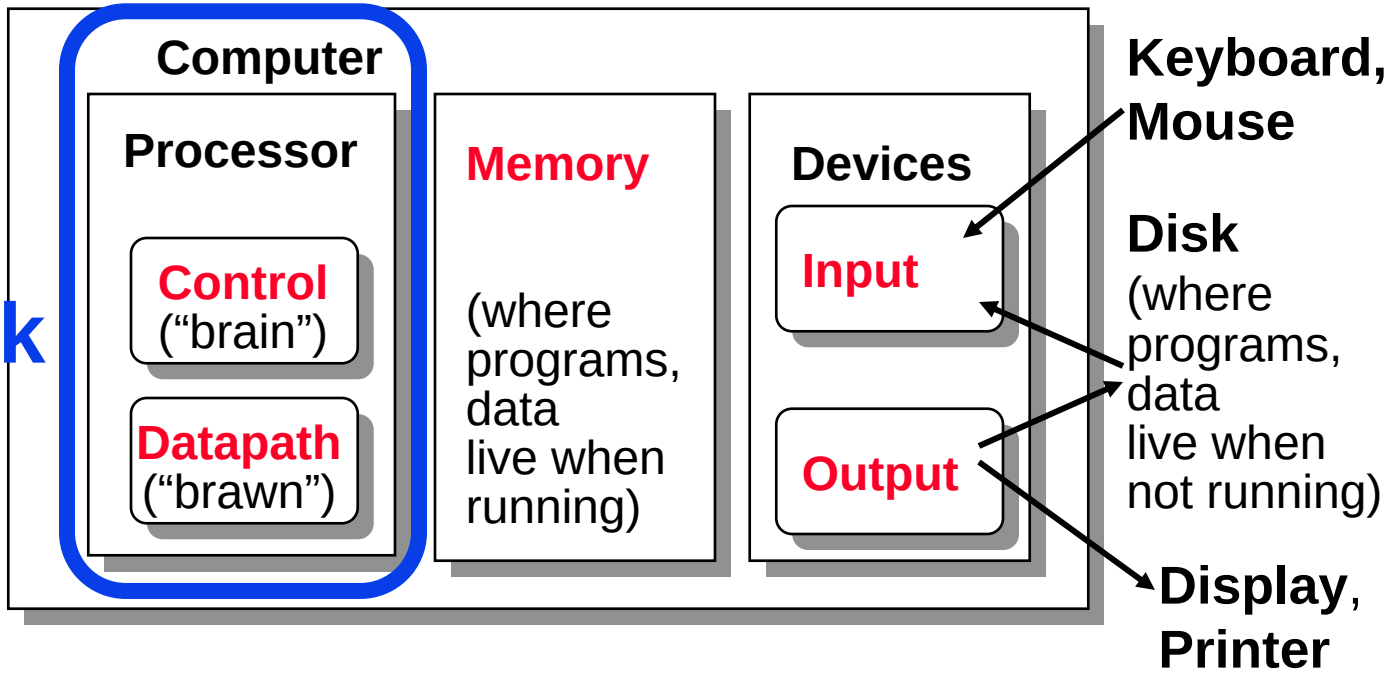
2006-07-25

Andy Carle

Anatomy: 5 components of any Computer



This week



Outline

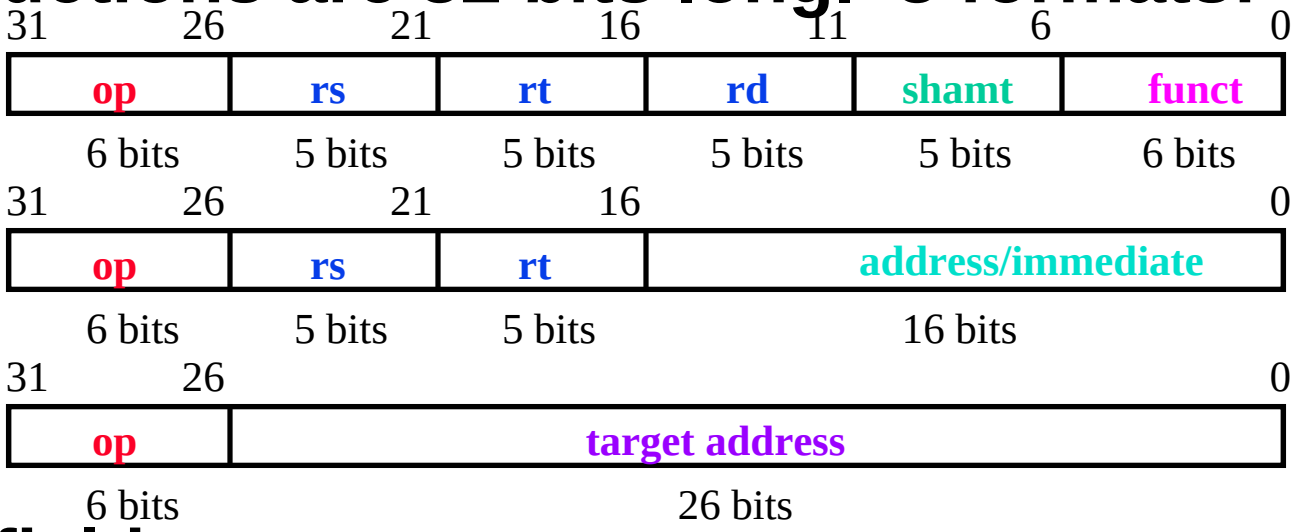
- Design a processor: step-by-step
- Requirements of the Instruction Set
- Hardware components that match the instruction set requirements

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

Step 1: The MIPS Instruction Formats

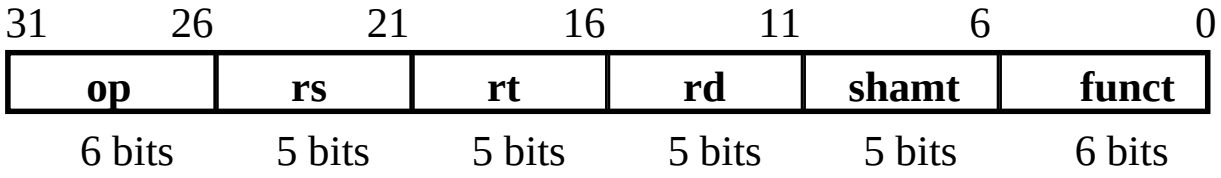
- All MIPS instructions are 32 bits long. 3 formats:
  - R-type
  - I-type
  - J-type



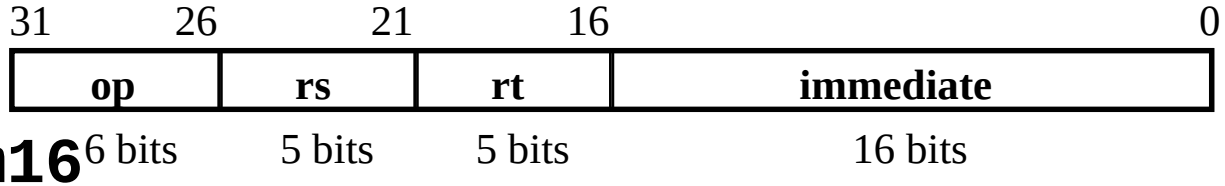
- The different fields are:
  - op: operation ("opcode") of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: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

Step 1: The MIPS-lite Subset for today

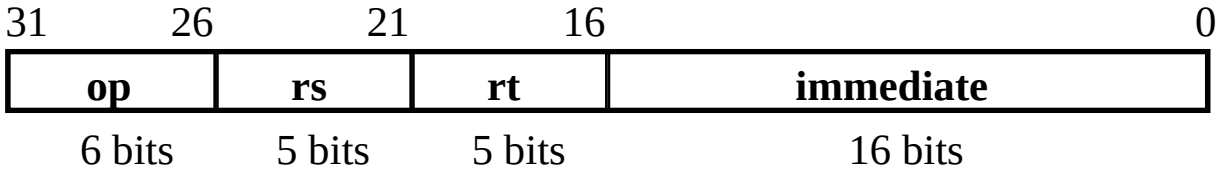
- **ADD and SUB**
  - `addU rd,rs,rt`
  - `subU rd,rs,rt`



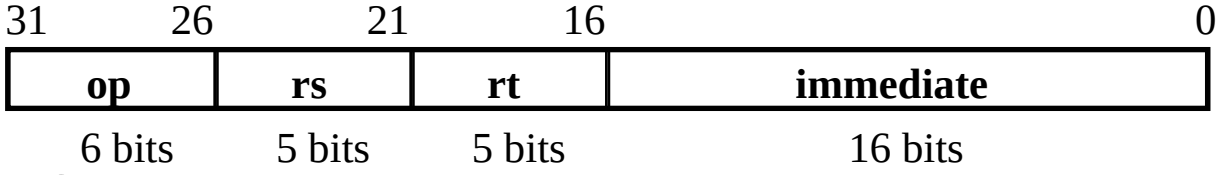
- **OR Immediate:**
  - `ori rt,rs,imm16`



- **LOAD and STORE Word**
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`



- **BRANCH:**
  - `beq rs,rt,imm16`



Step 1: Register Transfer Language

- **RTL gives the meaning of the instructions**

{op , rs , rt , rd , shamt , funct} = MEM[PC]

{op , rs , rt , Imm16} = MEM[PC]

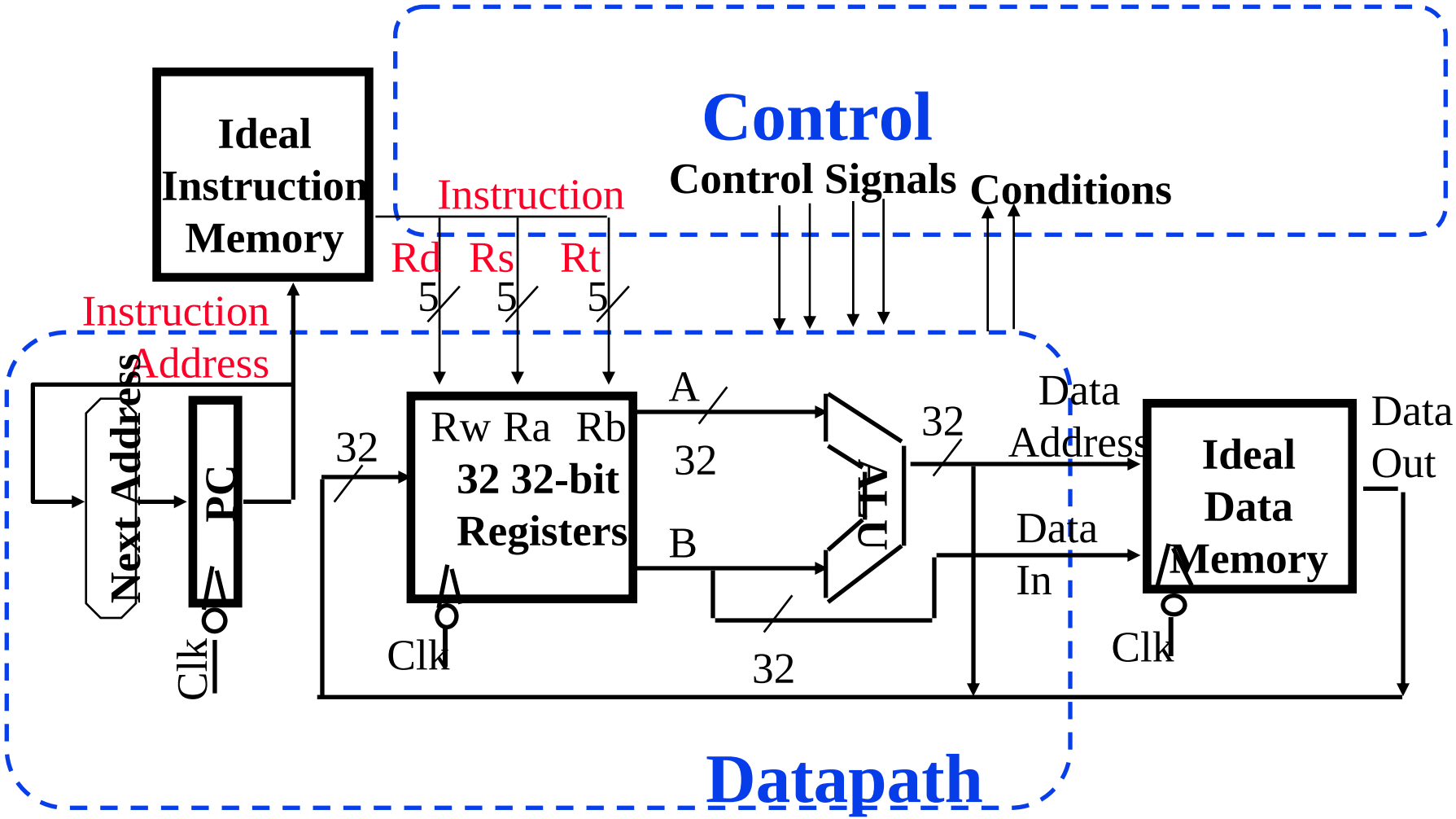
- **All start by fetching the instruction**

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 (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 1: Abstract Implementation



How to Design a Processor: step-by-step

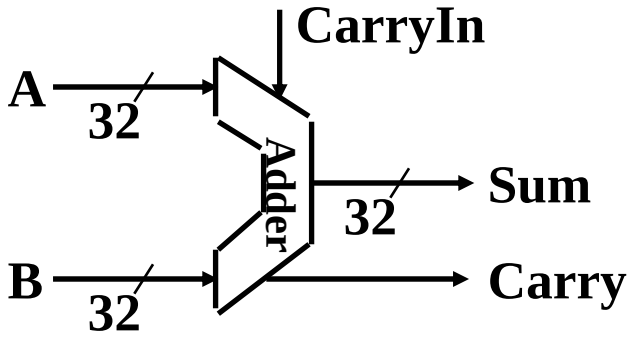
- 1. Analyze instruction set architecture (ISA) => datapath requirements
  - meaning of each instruction is given by the *register transfers*
  - datapath must include storage element for ISA registers
  - datapath must support each register transfer
- 2. Select set of datapath components and establish 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 (hard part!)

Step 2a: Components of the Datapath

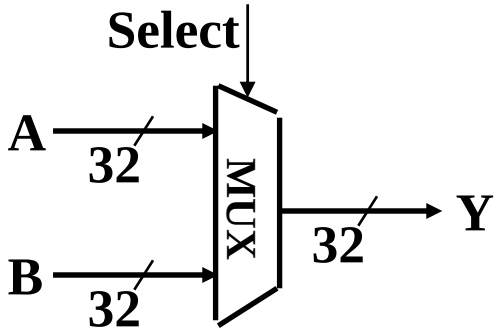
- **Combinational Elements**
- **Storage Elements**
  - **Clocking methodology**

Combinational Logic: More Elements

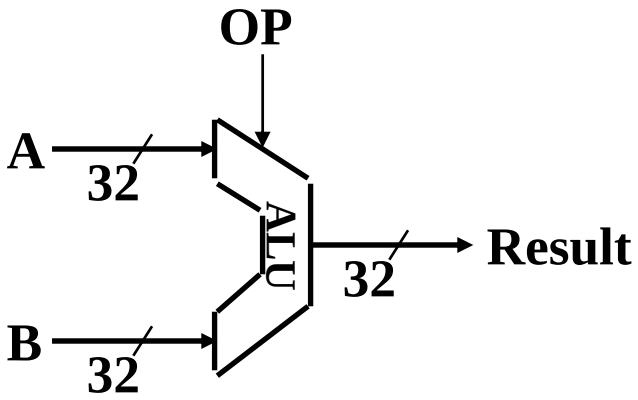
- **Adder**



- **MUX**



- **ALU**



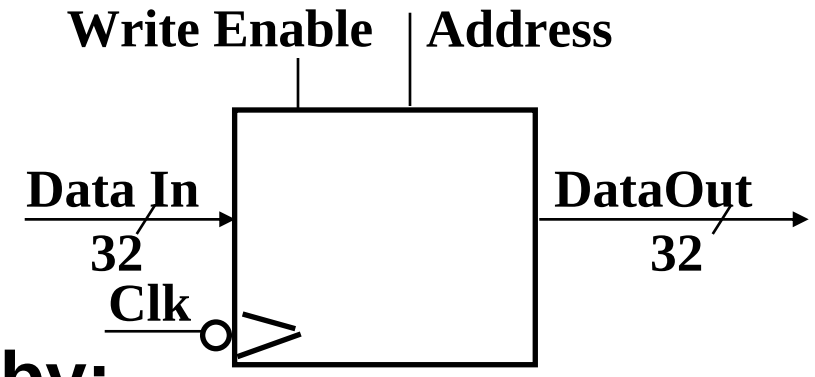
ALU Needs for MIPS-lite + Rest of MIPS

- 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 gives == test. How?
- P&H also adds AND,
  Set Less Than (1 if A < B, 0 otherwise)
- ALU follows chap 5

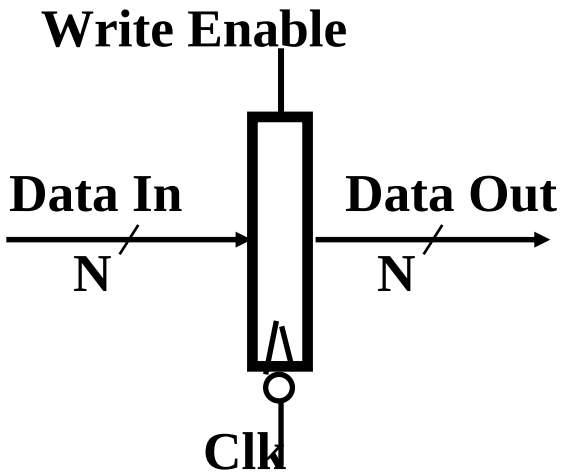
Storage Element: Idealized Memory



- 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, behaves as a combinational logic block:
    - Address valid => Data Out valid after "access time."

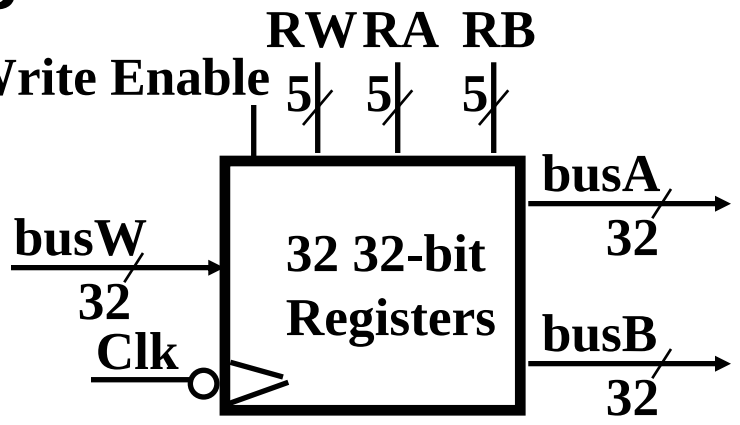
Storage Element: Register (Building Block)

- **Similar to D Flip Flop except**
  - **N-bit input and output**
  - **Write Enable input**
- **Write Enable:**
  - **negated (or deasserted) (0): Data Out will not change**
  - **asserted (1): Data Out will become Data In**



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."

Administrivia

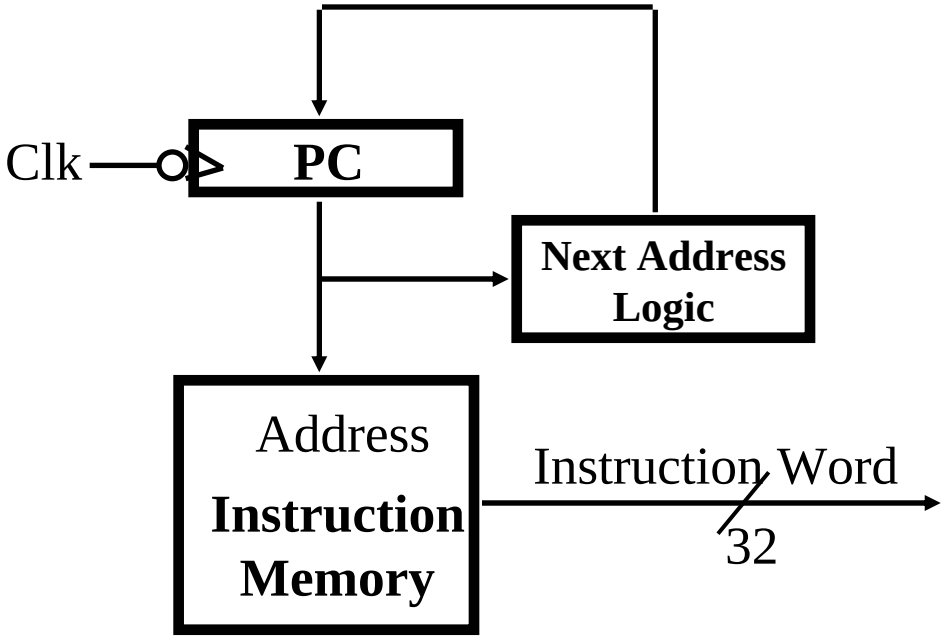
- Project 2 due Friday
  - Hope you've already started
- HW5 (maybe HW56?) out soon

Step 3: Assemble DataPath meeting requirements

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

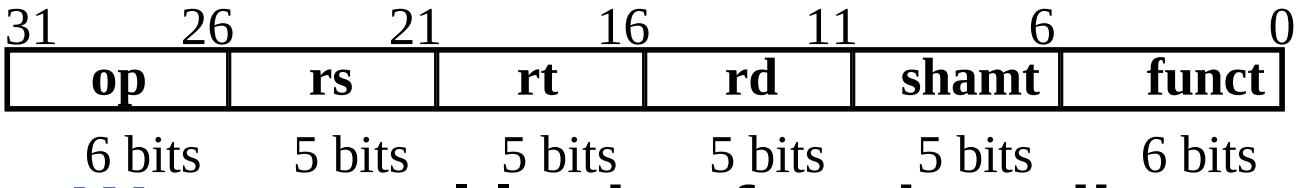
3a: Overview of the Instruction Fetch Unit

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

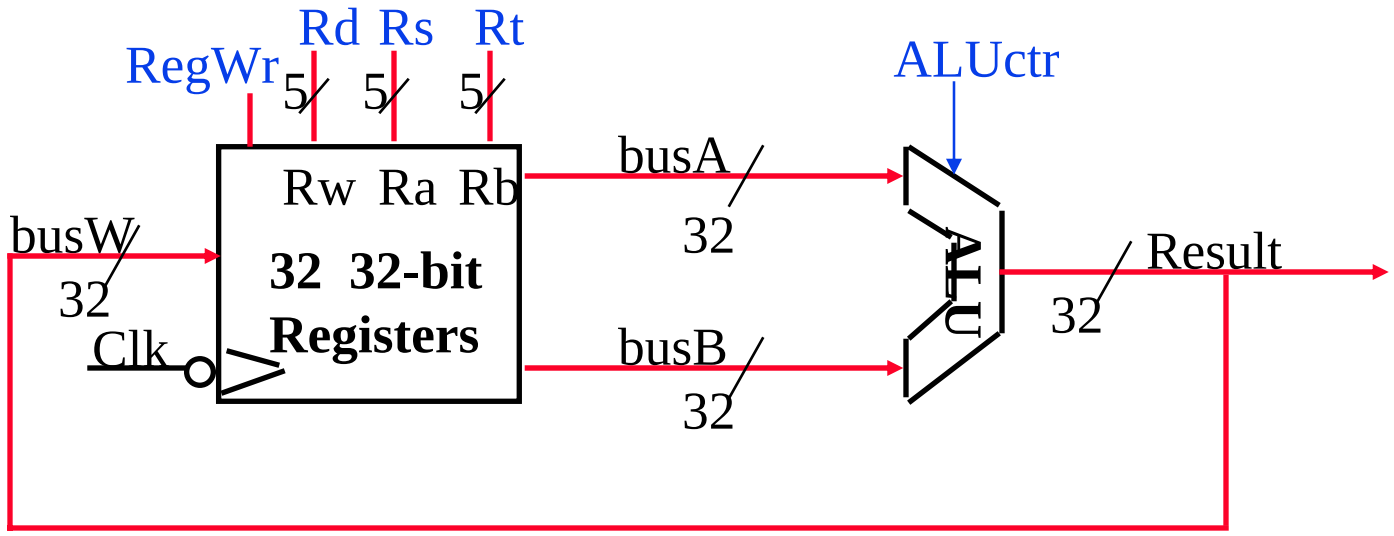


3b: Add & Subtract

- 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

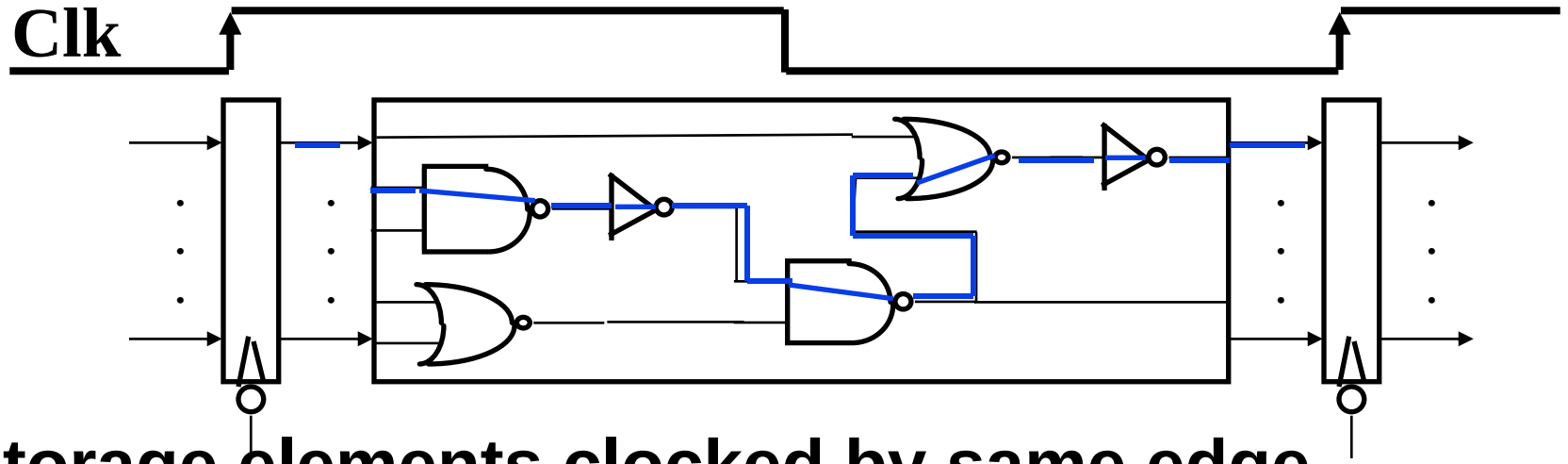


  - ALUctr and RegWr: control logic after decoding the instruction



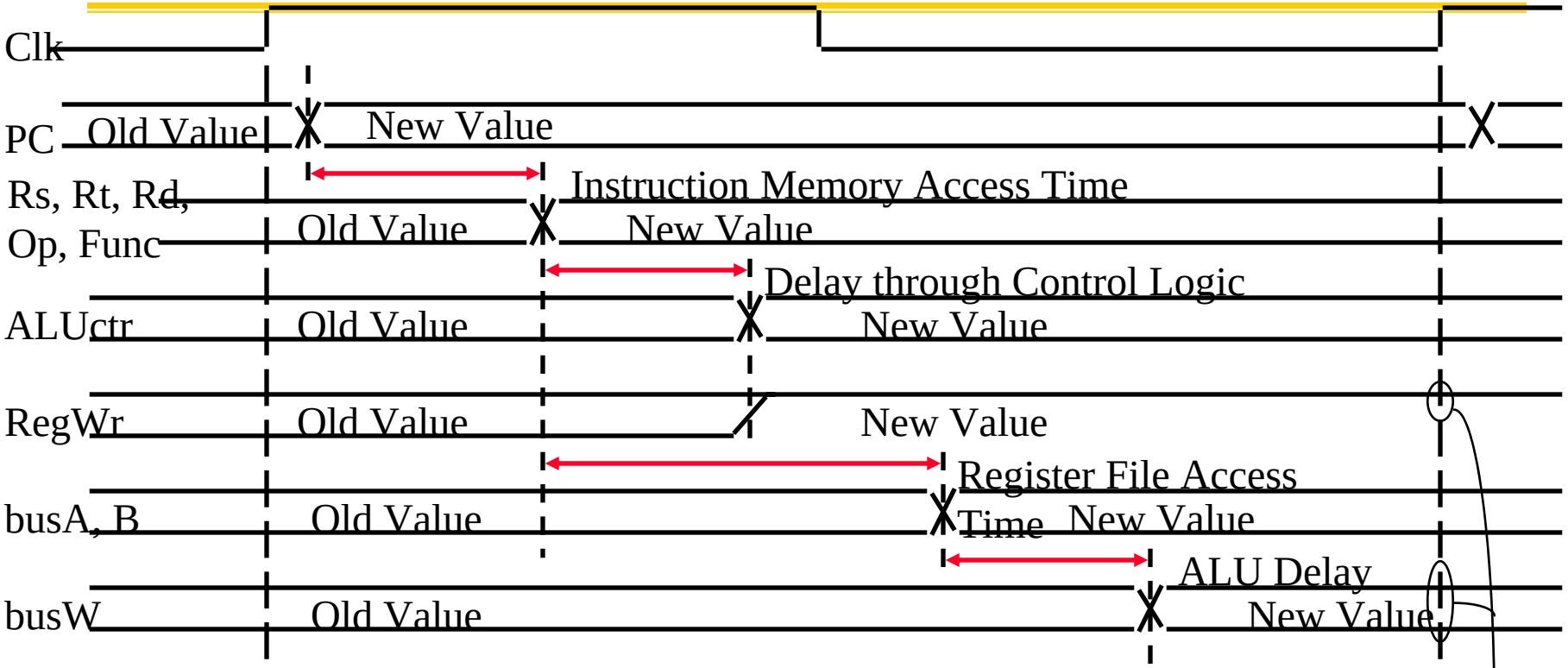
- Already defined register file, ALU

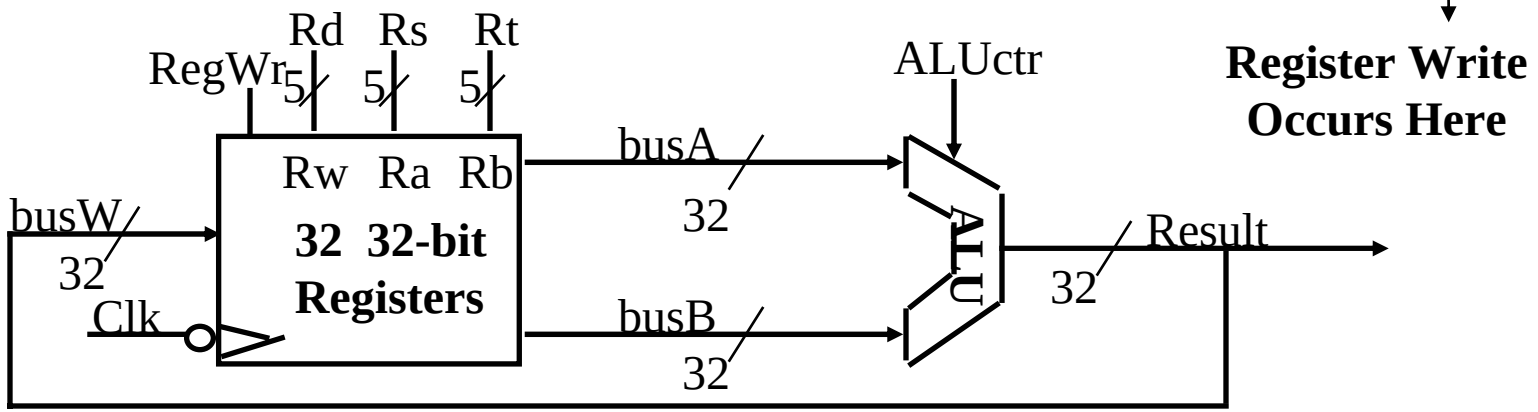
Clocking Methodology



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

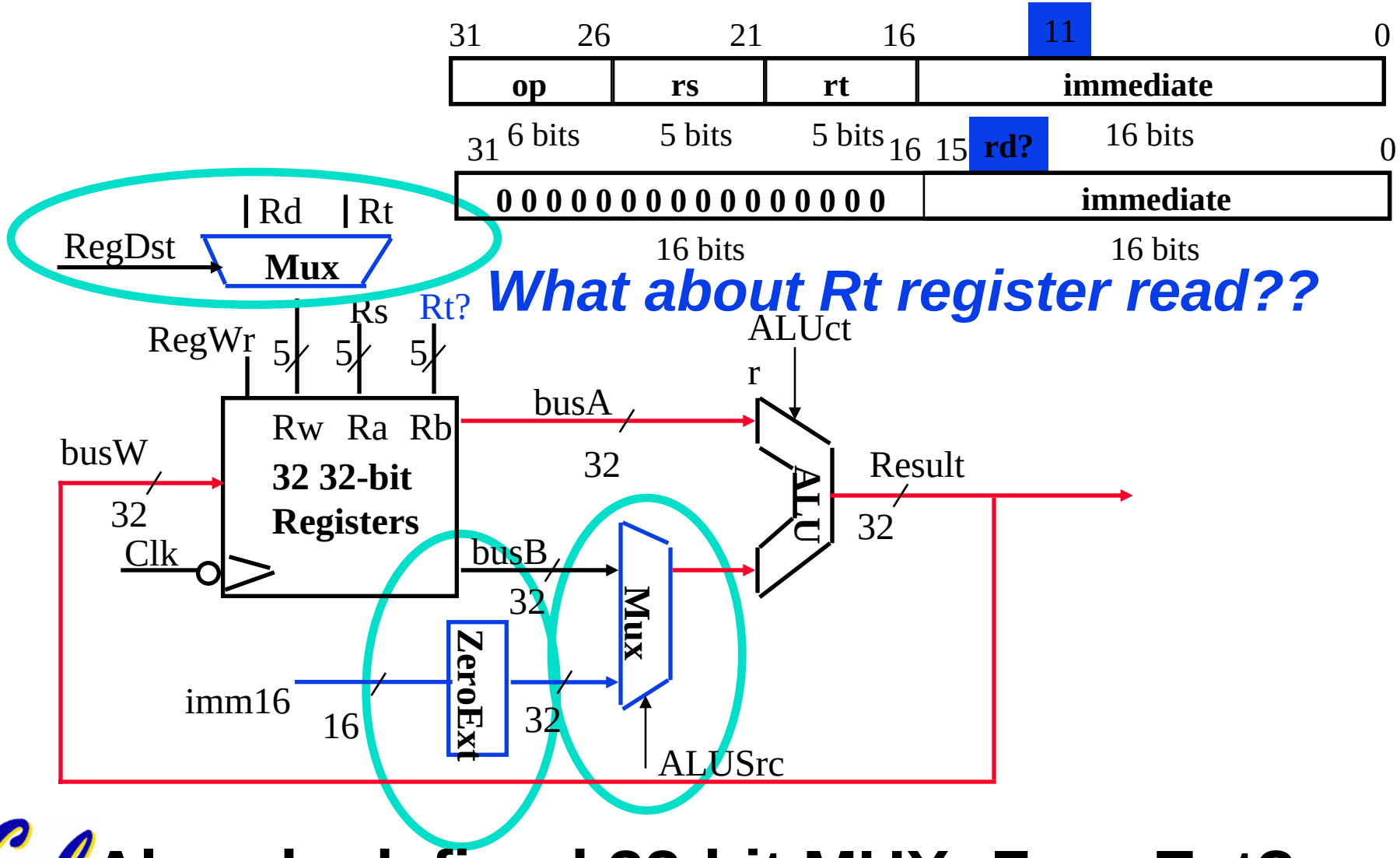
Register-Register Timing: One complete cycle





3c: Logical Operations with Immediate

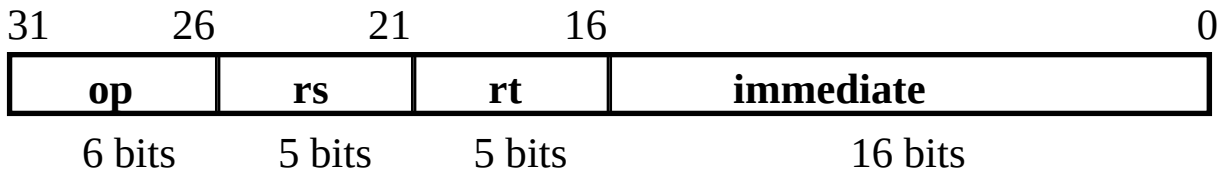
- R[rt] = R[rs] op ZeroExt[imm16]]

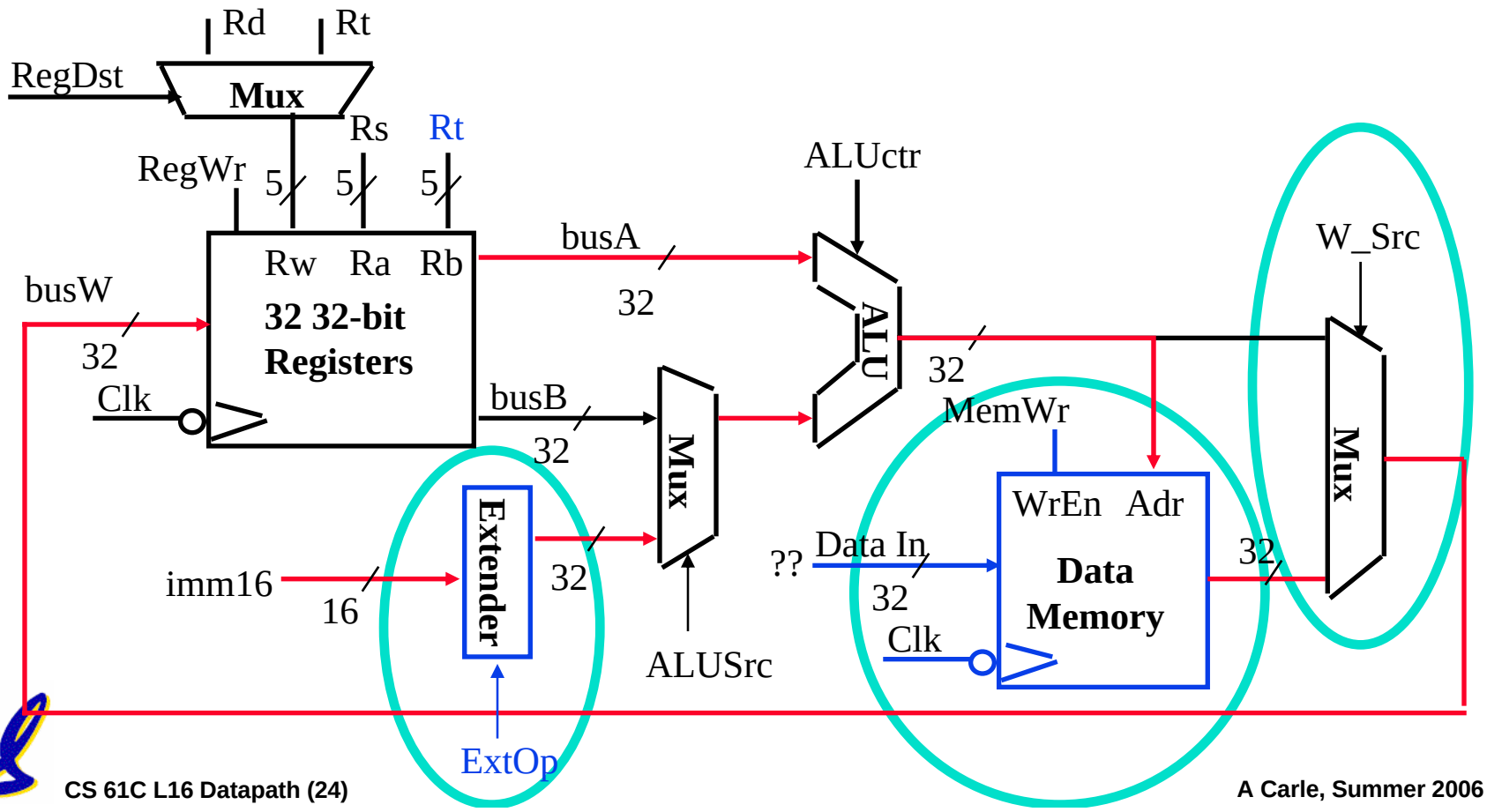


- Already defined 32-bit MUX; Zero Ext?

3d: Load Operations

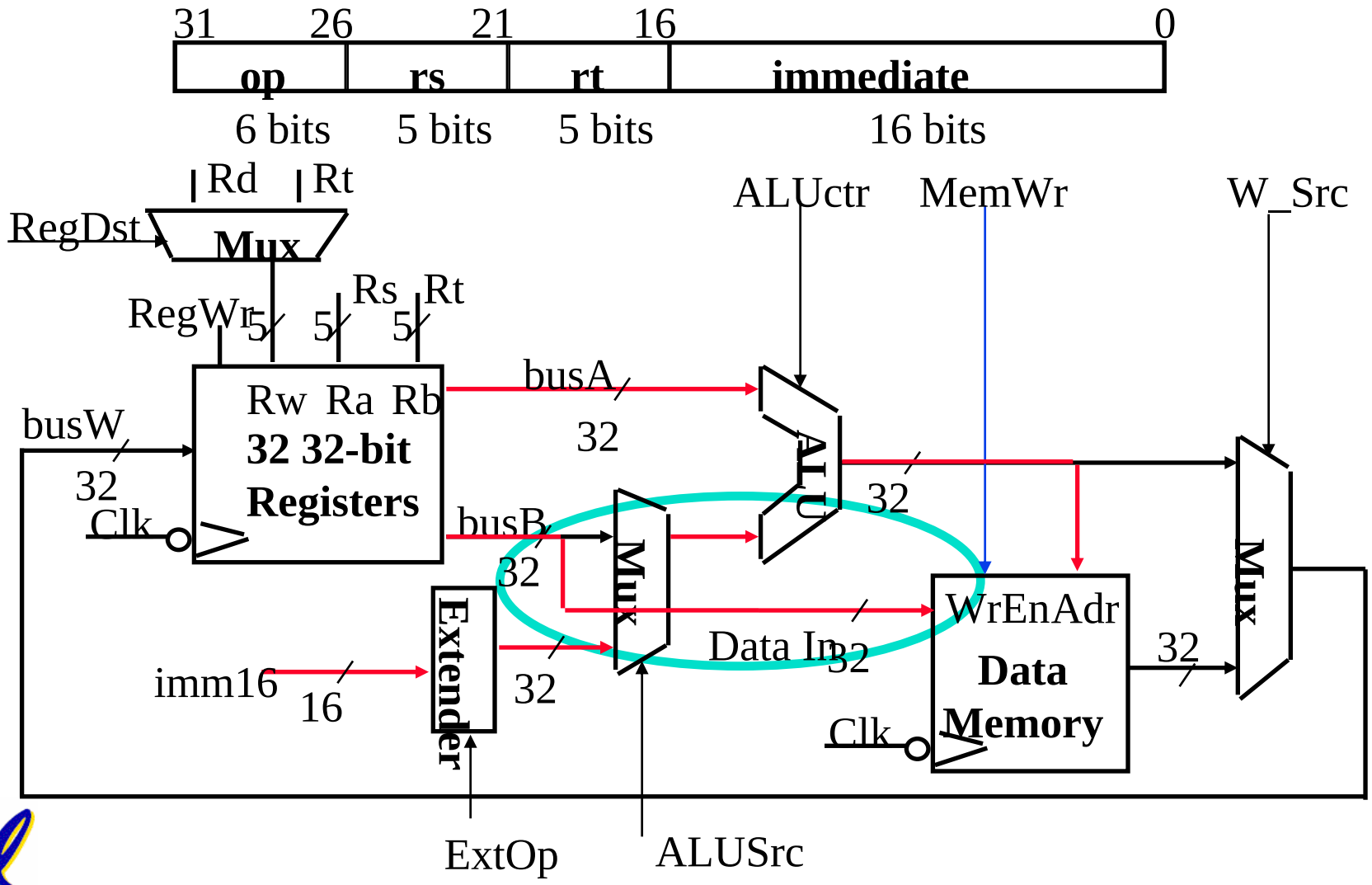
- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



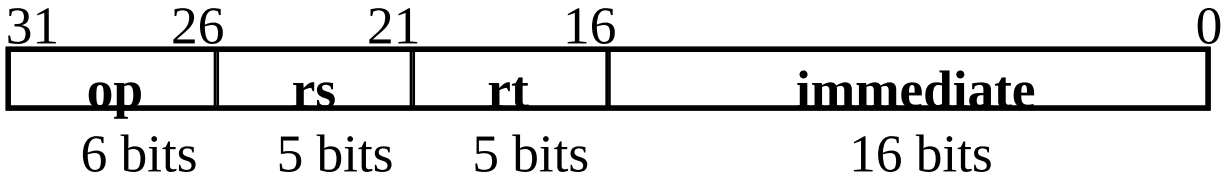


3e: Store Operations

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



3f: The Branch Instruction



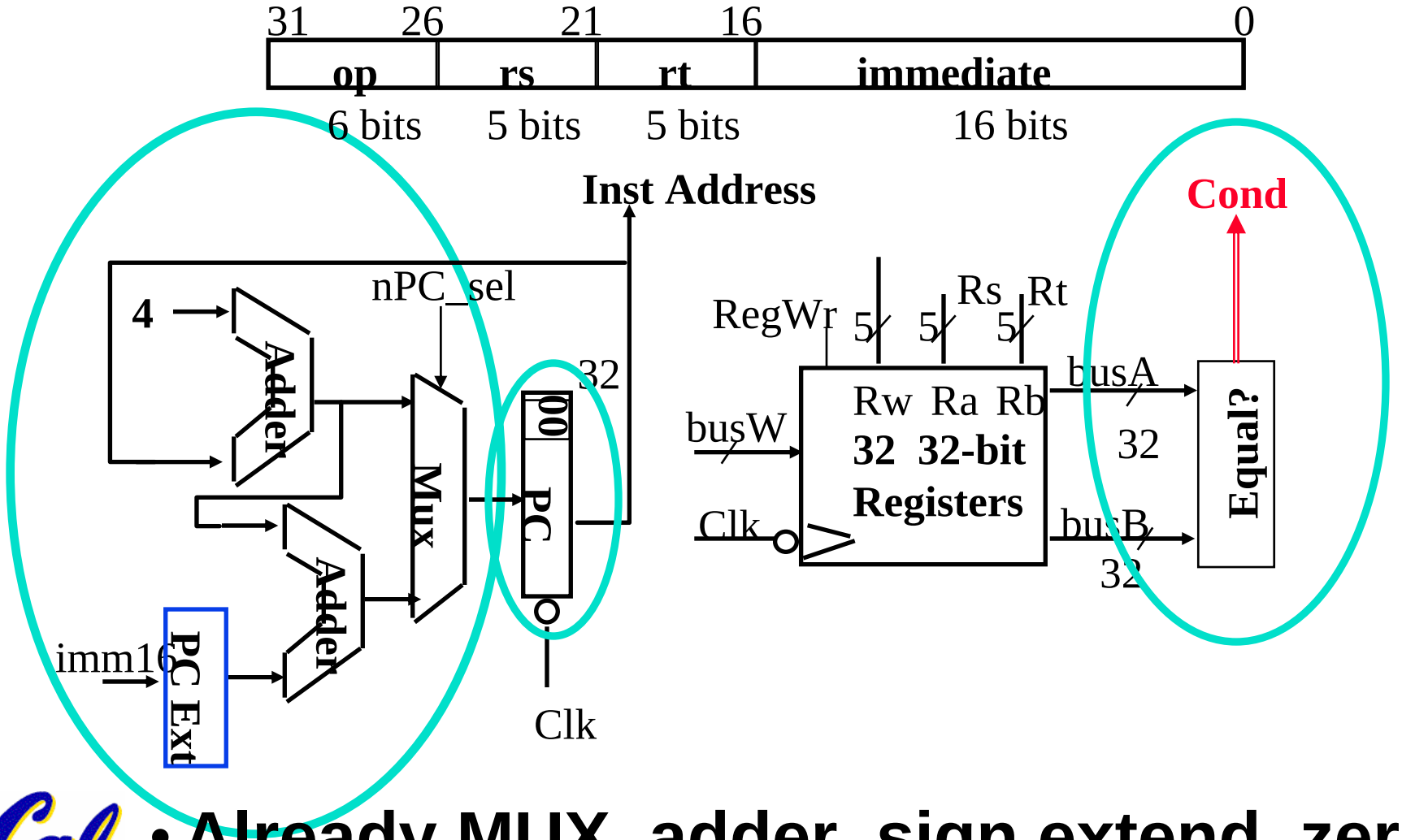
- **`beq rs, rt, imm16`**
  - **mem[PC] Fetch the instruction from memory**
  - **Equal = R[rs] == R[rt] Calculate branch condition**
  - **if (Equal) Calculate the next instruction's address**
    - **PC = PC + 4 + (SignExt(imm16) x 4)**

    else
    - **PC = PC + 4**

Datapath for Branch Operations

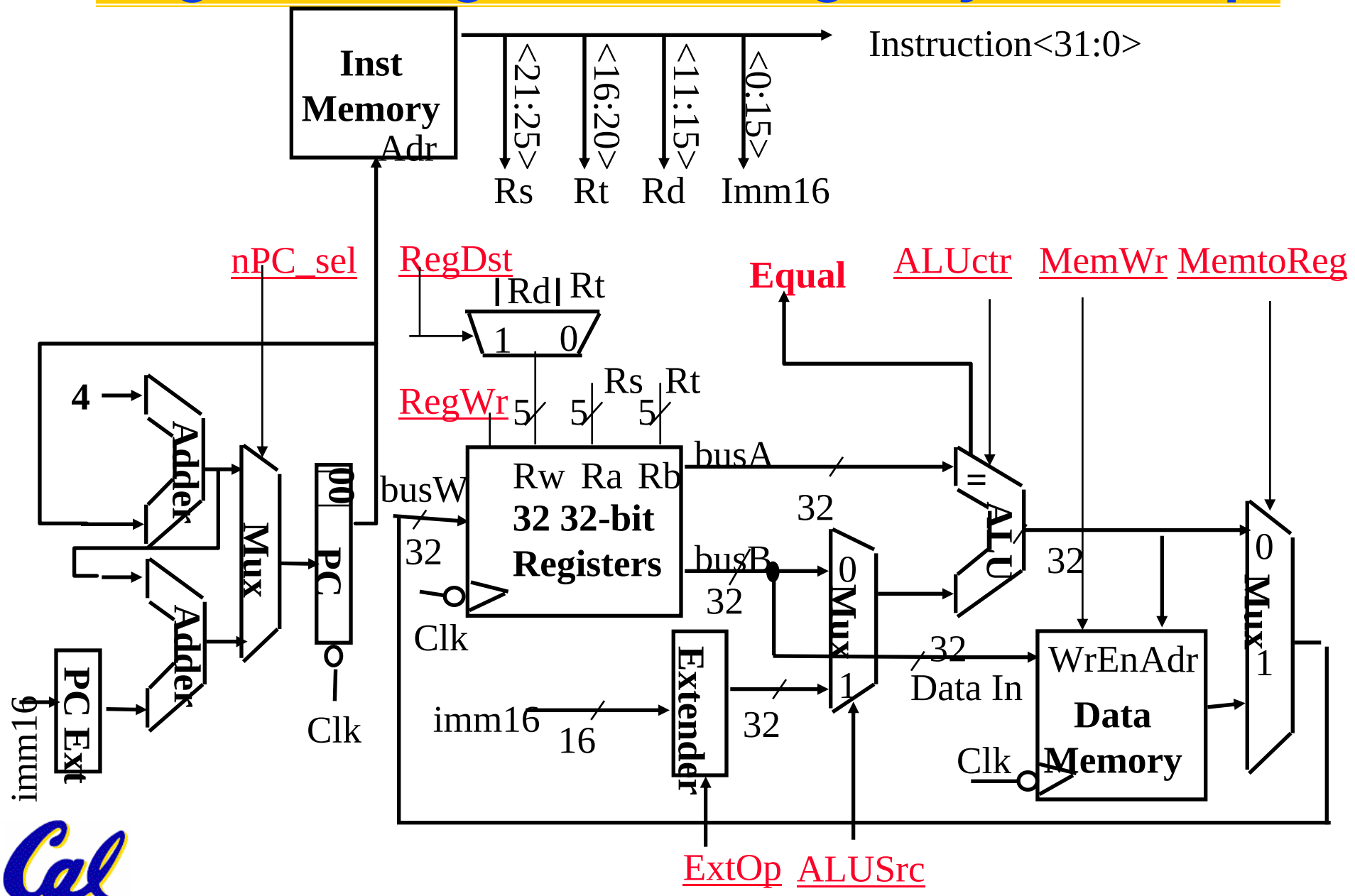
- **beq rs, rt, imm16**

Datapath generates condition (equal)



- **Already MUX, adder, sign extend, zero**

Putting it All Together:A Single Cycle Datapath



Peer Instruction

A. Our ALU is a synchronous device

B. We should use the main ALU to compute PC=PC+4

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

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!