CS3350B Computer Organization
Chapter 3: CPU Control & Datapath
Part 3: CPU Control

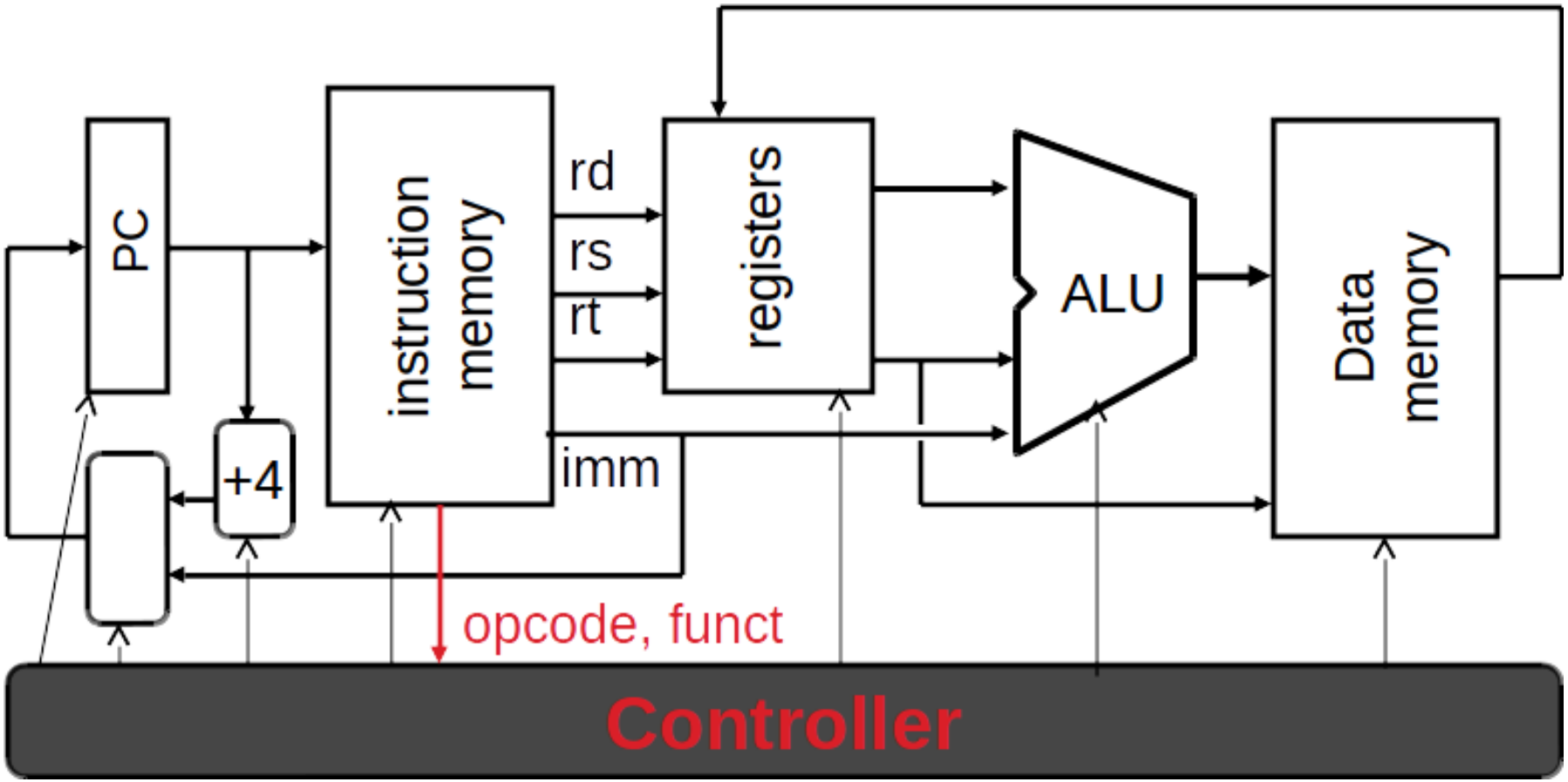
Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

Monday February 26, 2024

Outline

Controlling the Datapath



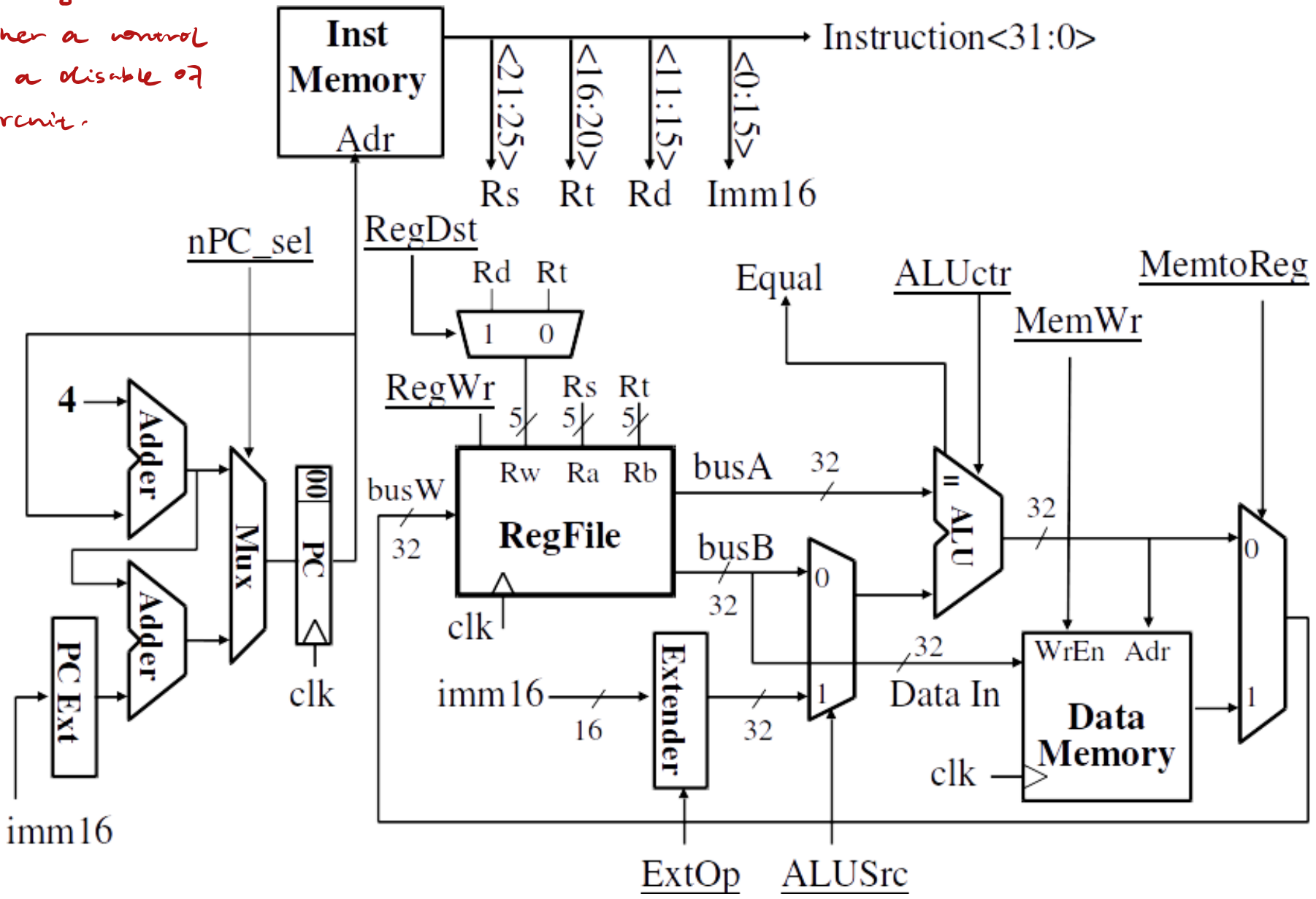
Control Signals

- Just as we saw with circuits like MUX, DEMUX, ALU, some circuits need **control signals** to help data flow or control the operation of the circuit.

- For an entire CPU datapath, this is called the **CPU controller**.
  - ↳ The controller contains the logic which interprets instructions and sends out control signals.
  - ↳ Many independent control signals are sent from the controller to each stage.
  - ↳ Sometimes multiple signals are sent to one stage, each controlling a different component within a stage.

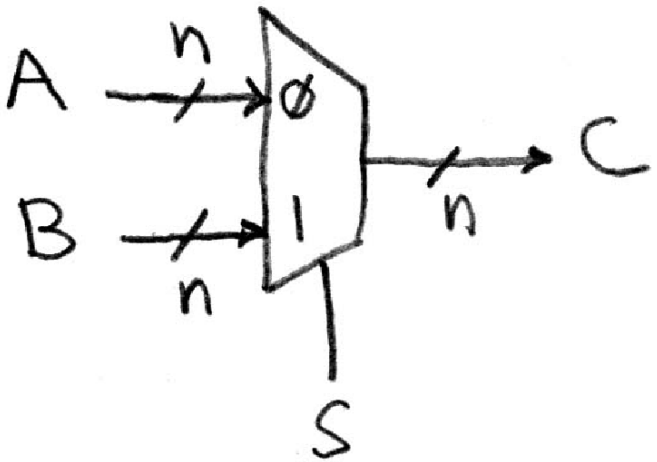
MIPS Datapath with Control Signals

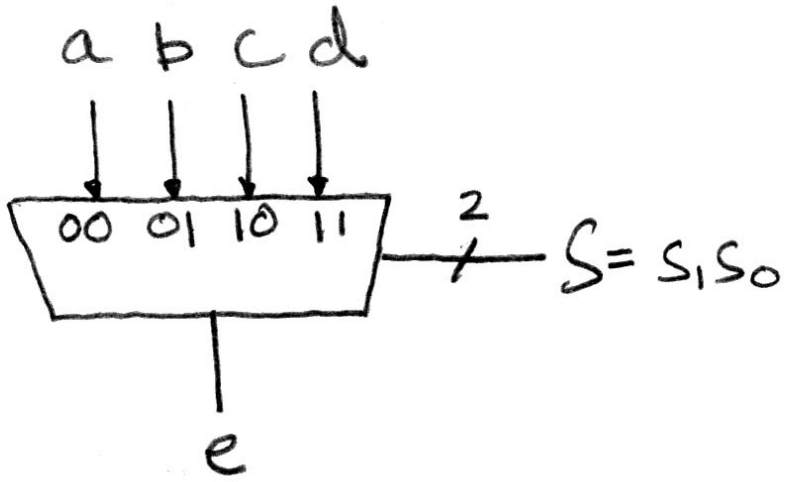
& ctrl signs.
=> either a control or a disable of circuit.



Outline

Review: MUX



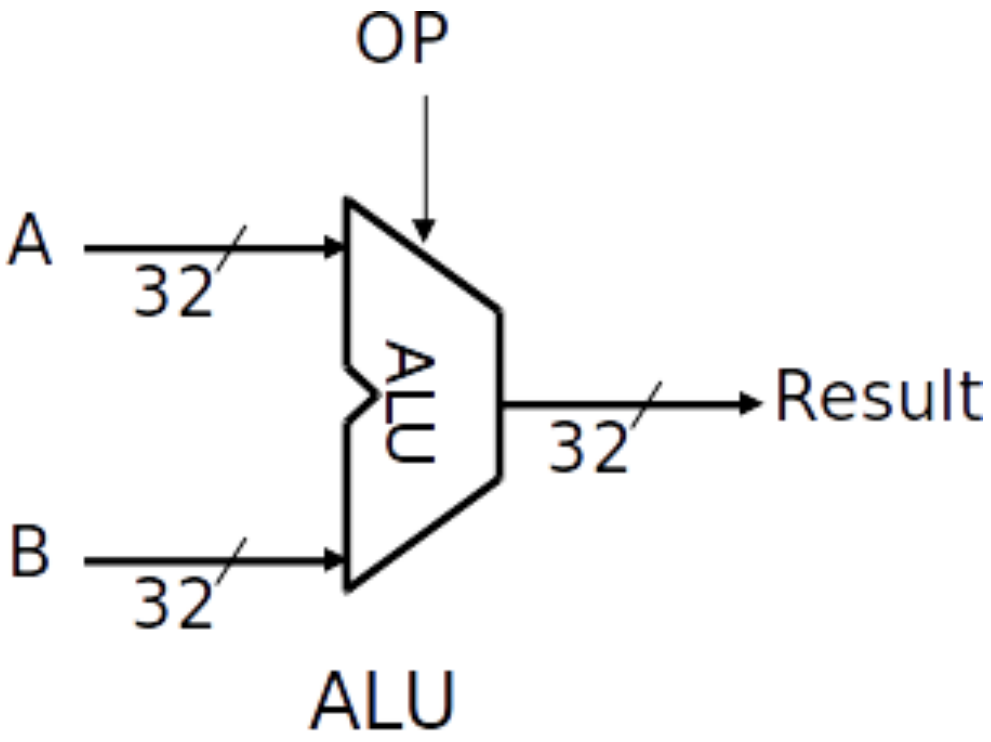


does not change of inputs; selection only.

The control signal **S** determines which input is used to set the output.

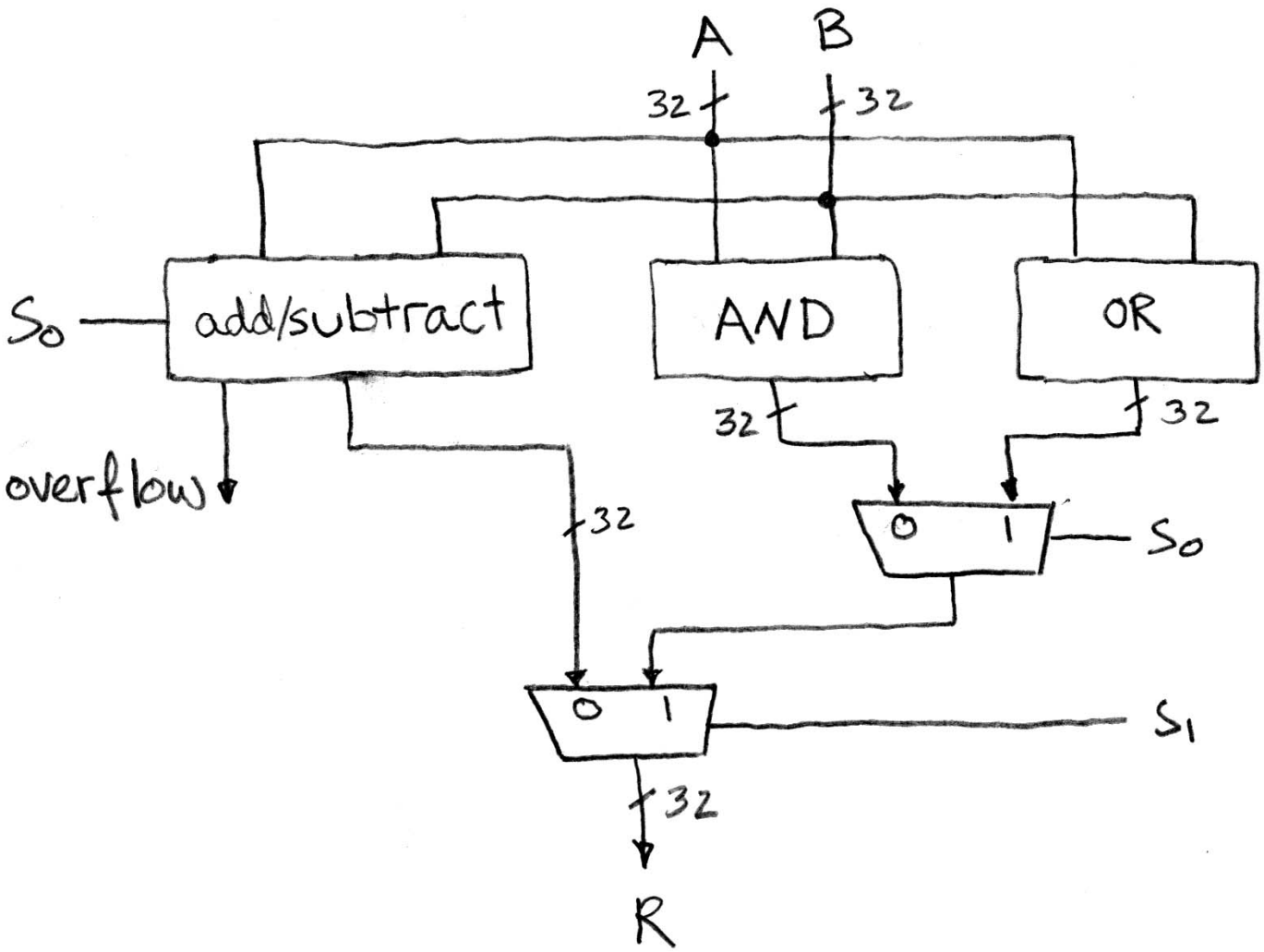
- Controls the flow of data.
- Bit-width of control signal determined by *number* of inputs to choose between, not the bit-width of the input.

Review: ALU



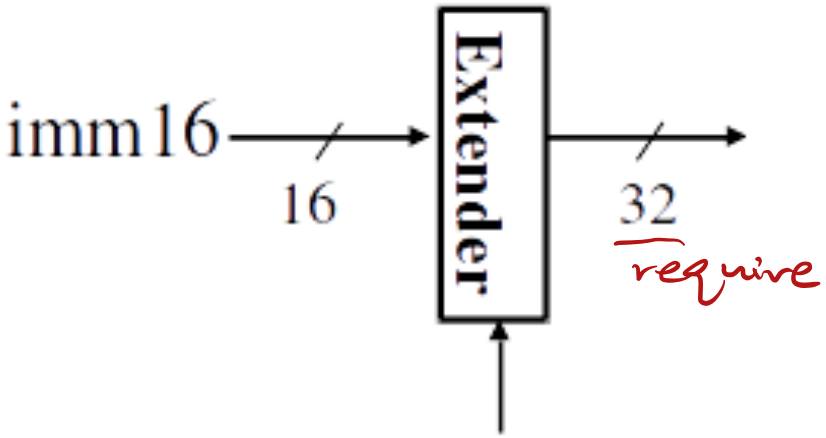
The control signal **OP** determines which arithmetic or logical operation is actually performed.

Review: ALU Implementation



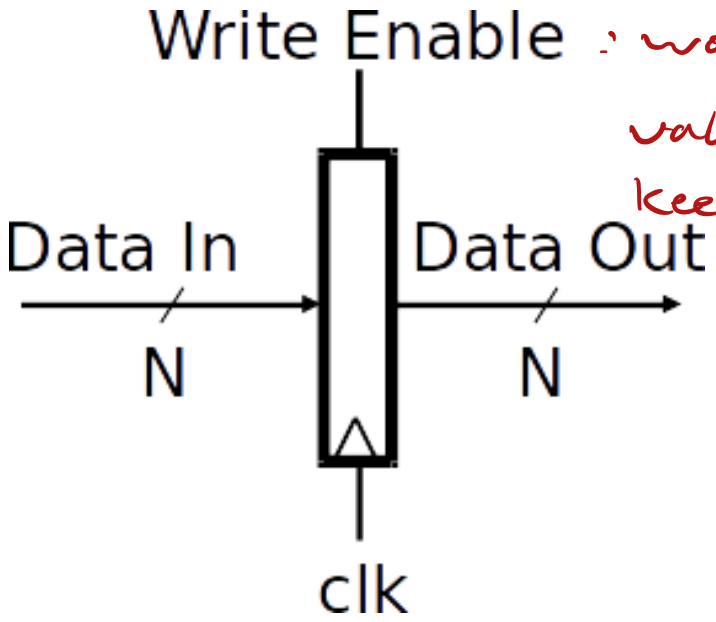
One possible ALU implementation. Do all of the operations, and control signal just controls a MUX to output.

Controlling Number Extenders



- **Extender:** A circuit which extends the bit-width of wire while maintaining its numerical value.
- *Recall:* we have both unsigned and signed numbers.
- Need a control signal to determine which to perform: `ExtOp`.

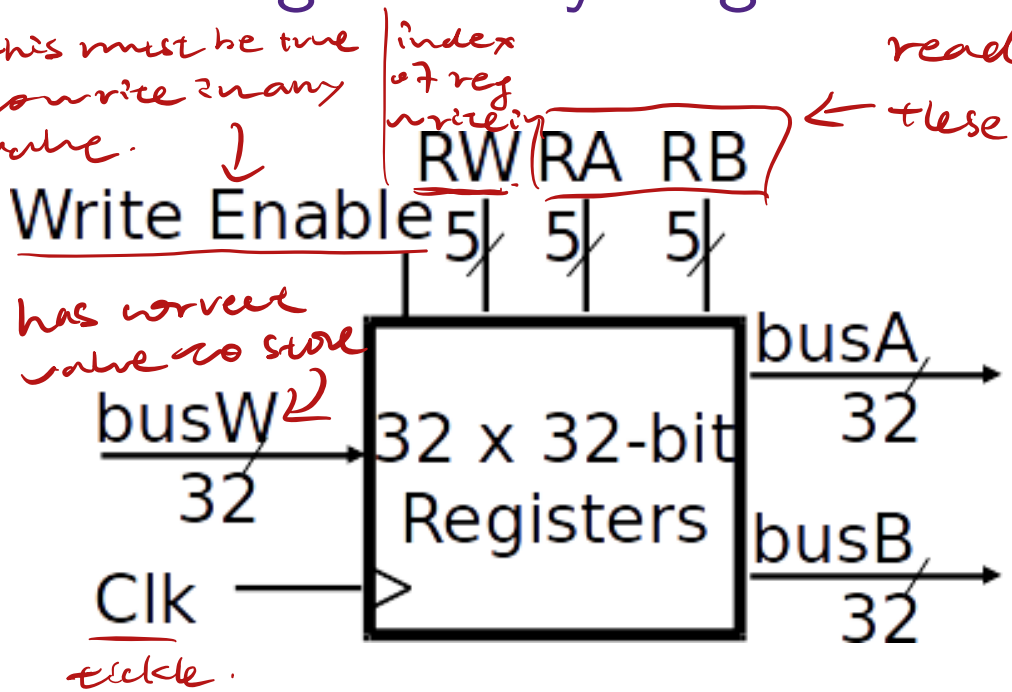
Controlling Data Storage: Register



: would register update value or ignore input and keep current state.

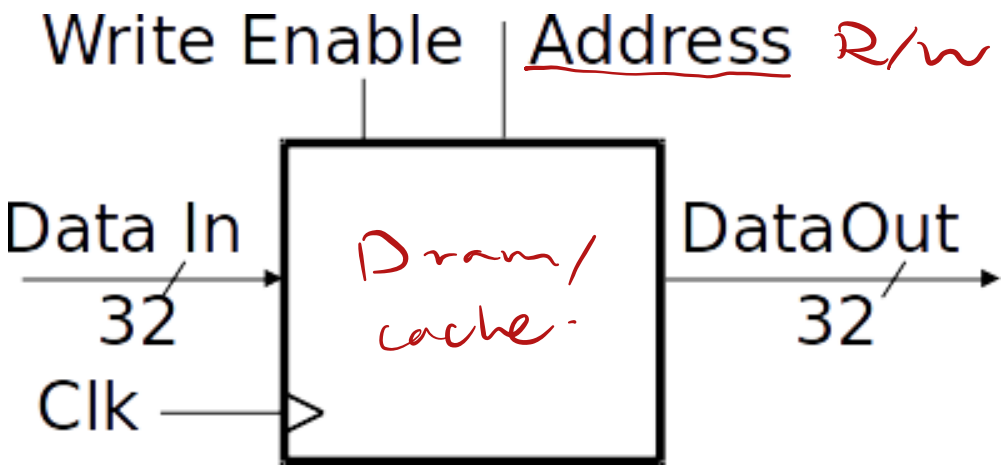
- Normally, registers are controlled by the clock.
- But, we can have special registers whose states are only updated when a special control signal is activated.
- These registers are updated when the control signal is 1 and the clock tic occurs simultaneously.

Controlling Data Storage: Many Registers



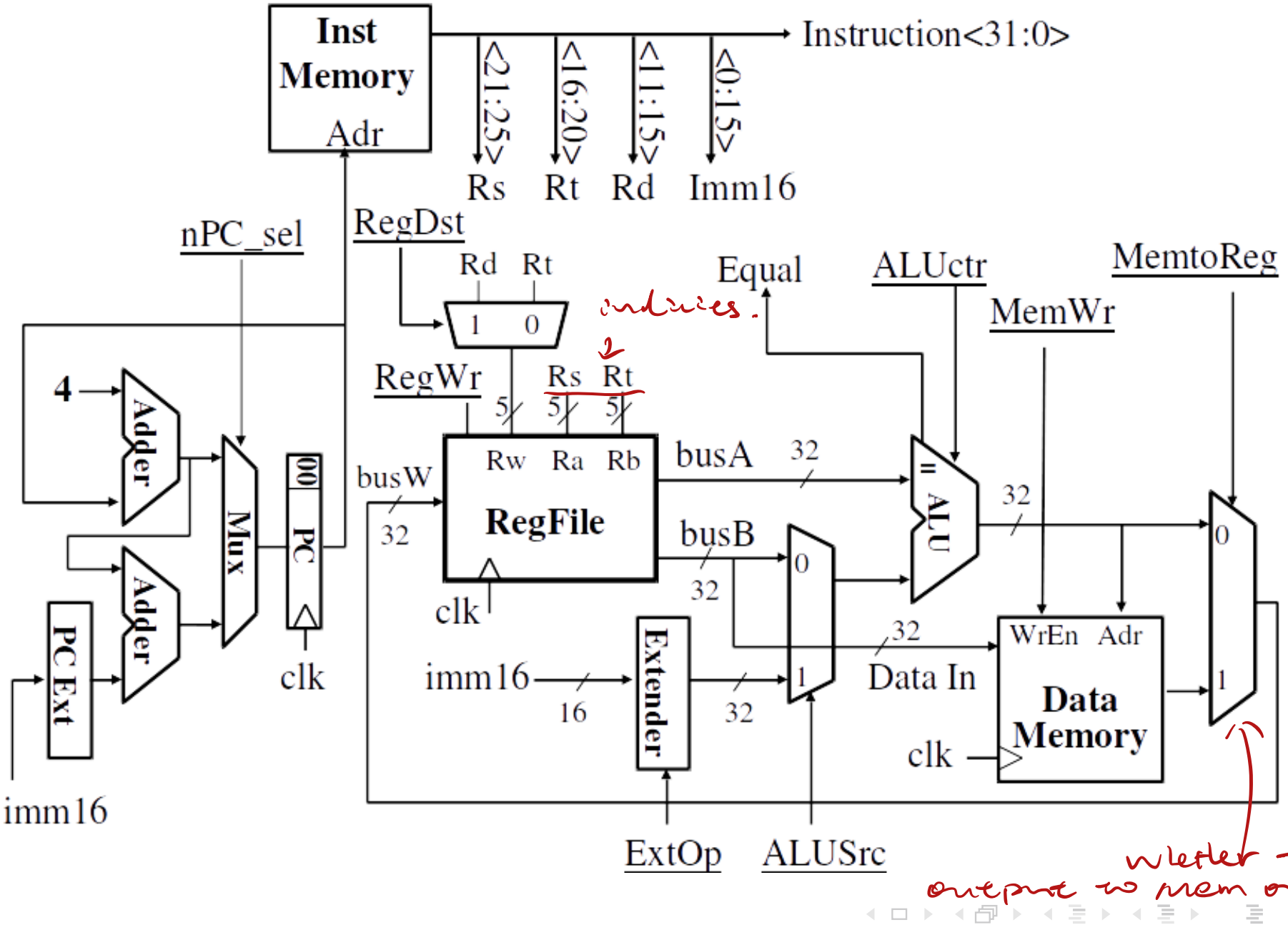
- A **register file** is a collection of registers put together.
- `RA` and `RB` are the *indices* of the registers we want to read from.
- `RW` is the *index* of the register we want to write to.
  - ↳ On the clock, **if** write enable control signal is 1, then write the data on `busW` to register `RW`.
- Clock does not affect *reads*, only *writes*.

Controlling Data Storage: Data Memory



- A simplified **data memory** works much like a register file.
- `Address` specifies the memory address to read from or write to.
- `DataOut` is the data read from memory.
- `DataIn` is the data to be written.
- A write only occurs on the clock tic and when `WriteEnable` is 1.
- Clock does not affect reads.

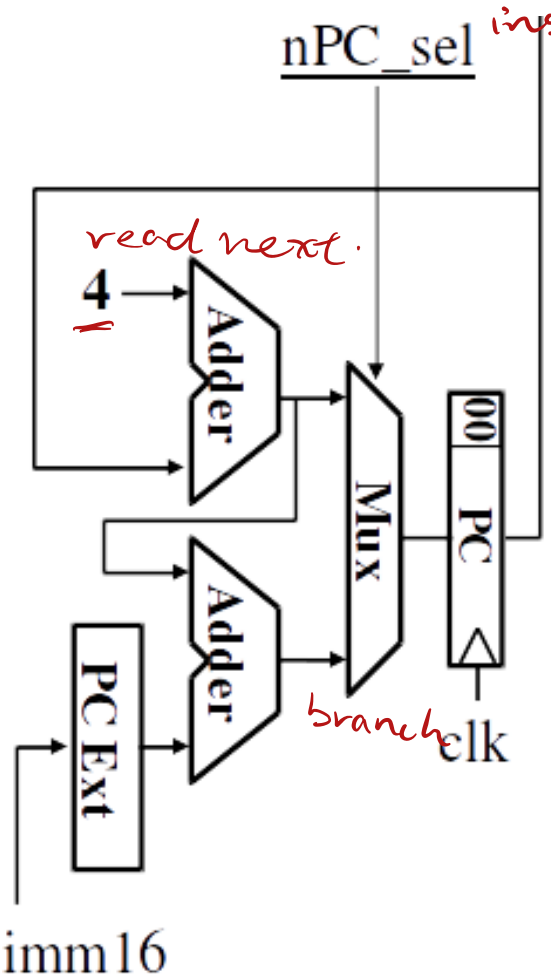
MIPS with Control Signals



Outline

Controlling "Instruction Fetch Unit" and PC

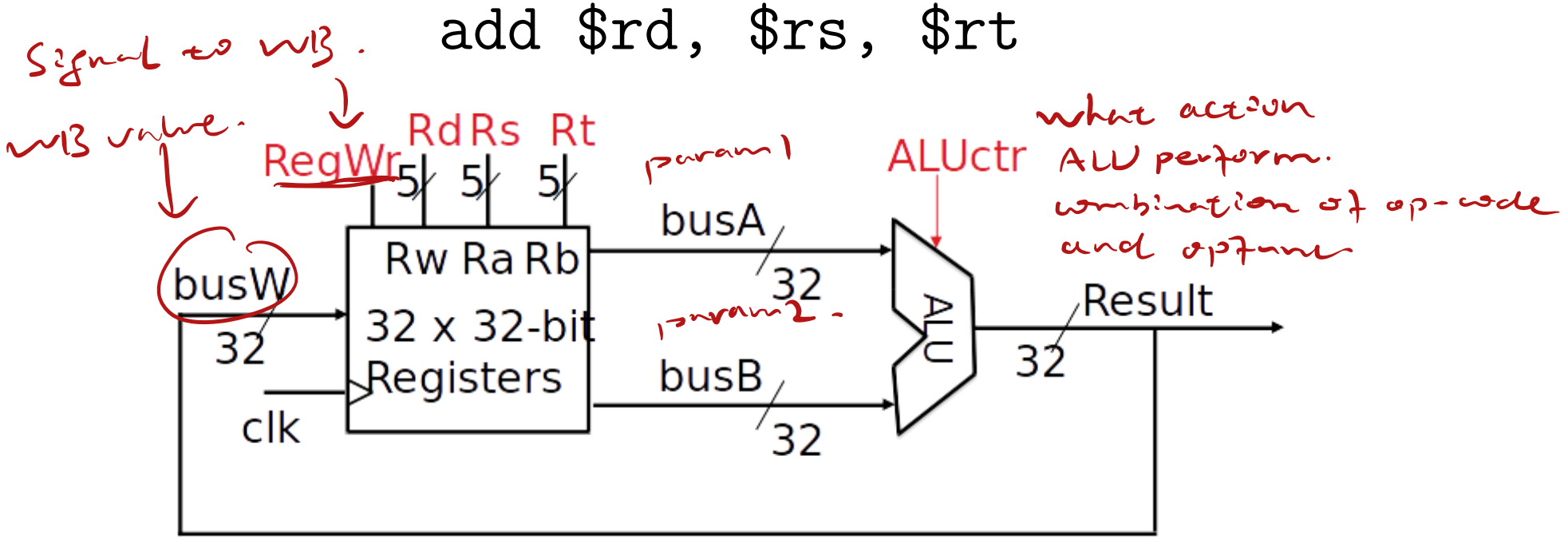
- For most instructions simply perform `PC = PC + 4`.
- For `branch` inst. we must do a special extension of the immediate value and then add it to PC..
- The next PC is actually decided by MUX and the `nPC_sel` control signal.
- If the branch condition evaluates to true, then the control signal is set to 1 and the MUX chooses the branch address.



Recall: on branch instructions `PC = PC + 4 + (imm << 2)`.
The + 4 will become clear in next chapter.

Tracing add (1/2)

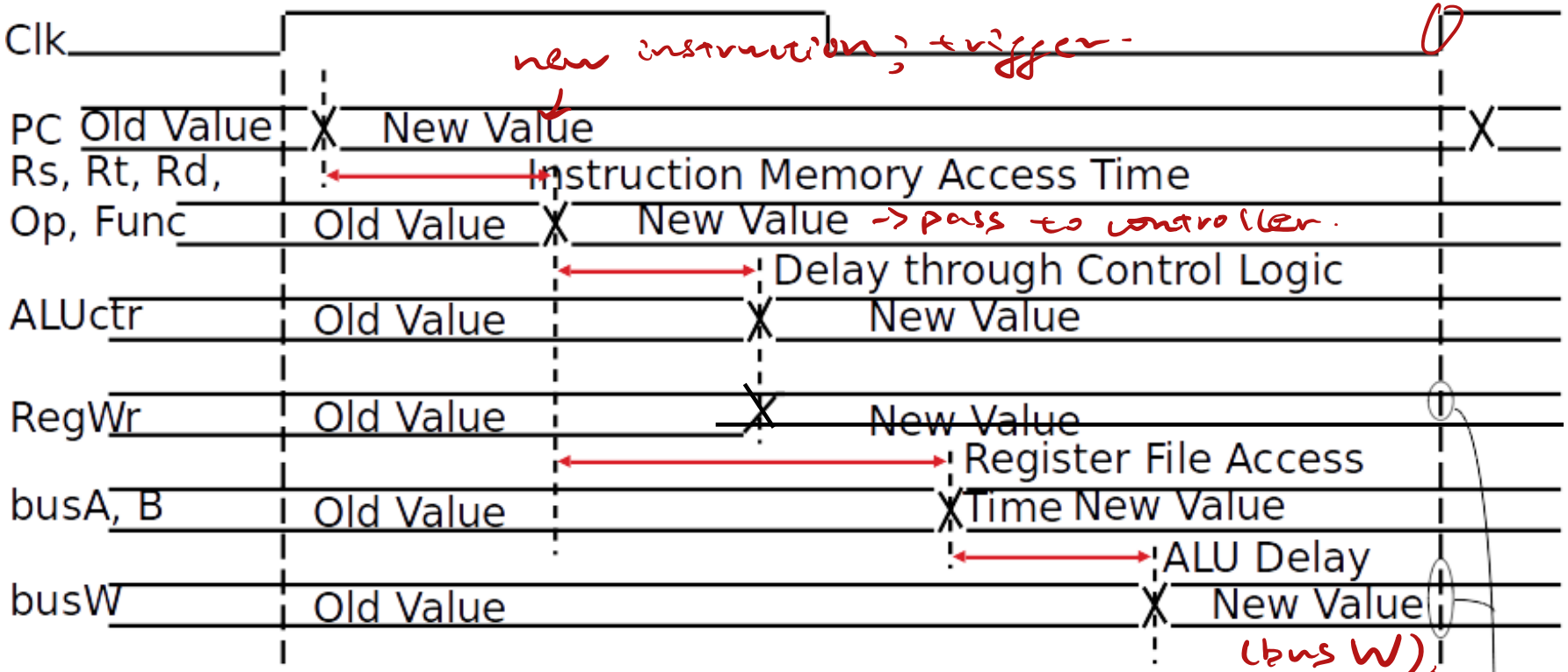
```
add $rd, $rs, $rt
```

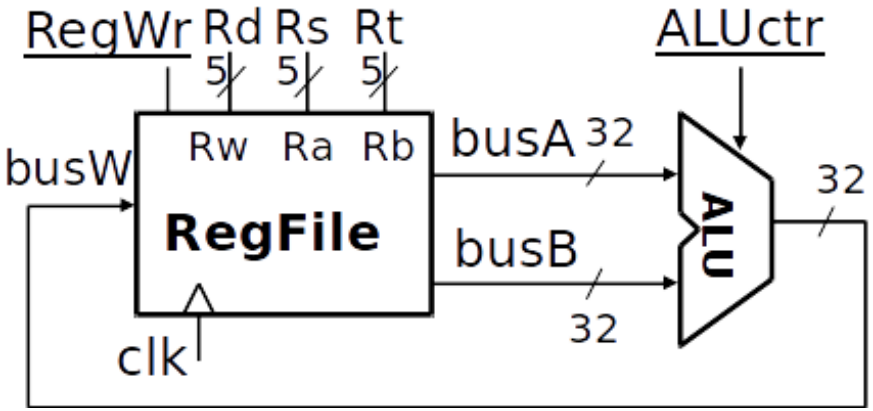


- Inst. *writes* to a register so the `RegWr` control signal must be true.
- The `ALUctr` signal is decided from the instruction ⇒ op and `funct`.
- add, addu, sub, subu, or, and, . . . , all have opcode = 0 but a different `funct`.

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

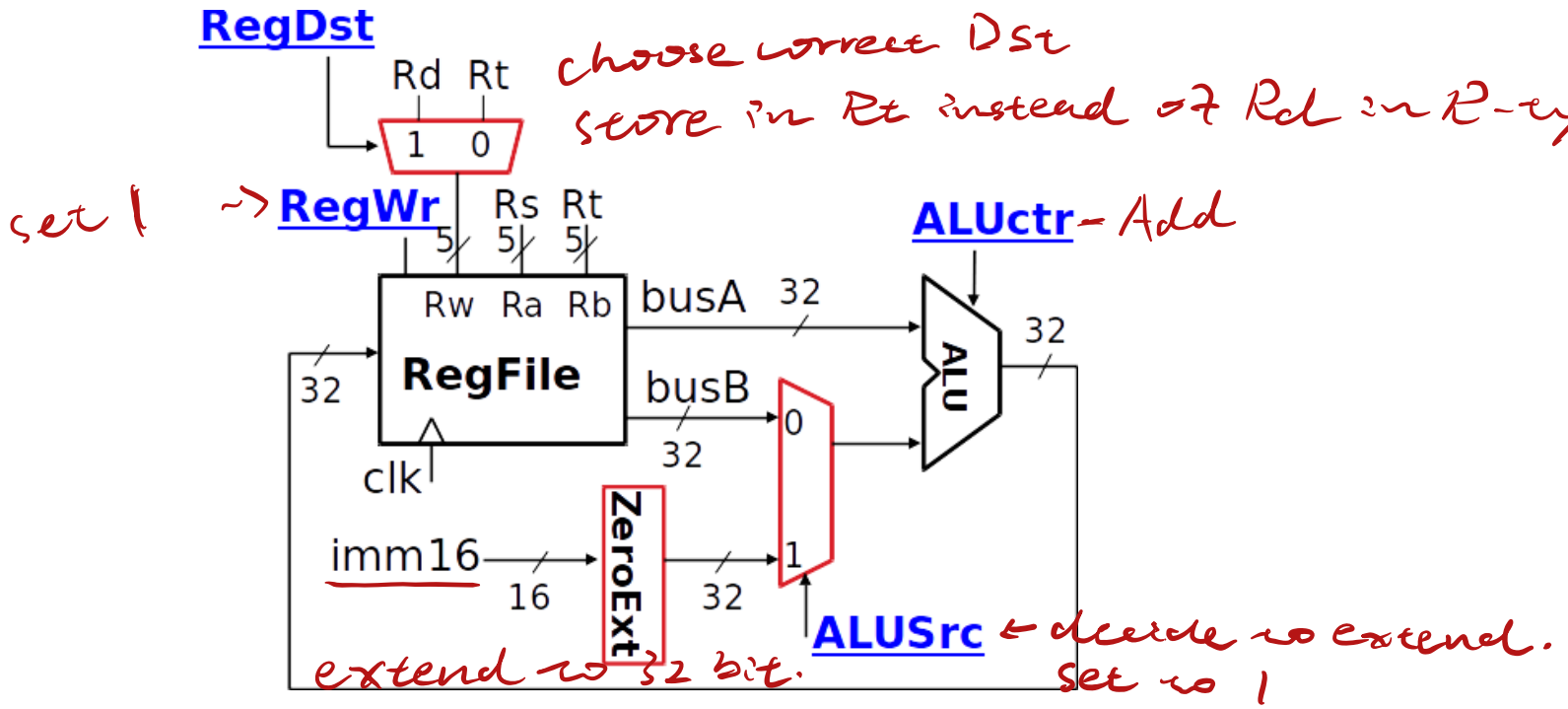
Tracing add (2/2)





Tracing addui

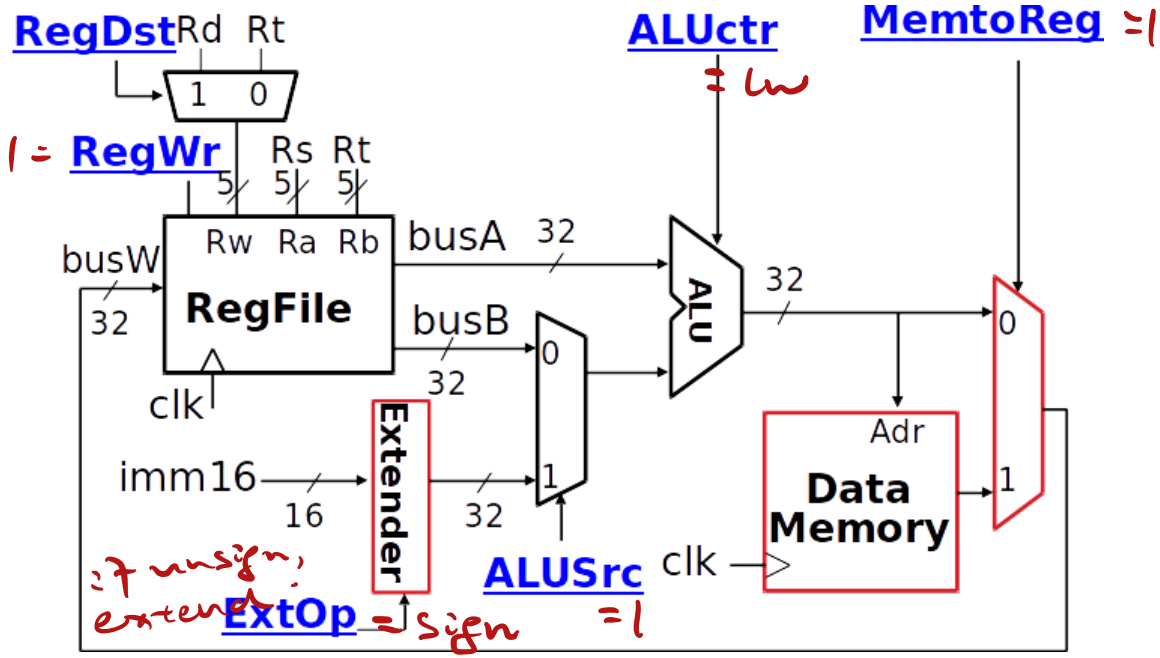
addui $rt, $rs, imm



- Modify previous path to allow register or immediate as input to ALU.
- Modify previous path to allow write to `rd` or `rt`.
- R-type inst. have `RegDst` = 'rd'; I-type have `RegDst` = 'rt'.
- R-type have `ALUSrc` = 'rt' or 'busB'; I-type have `ALUSrc` = 'imm.'.

Tracing `lw`

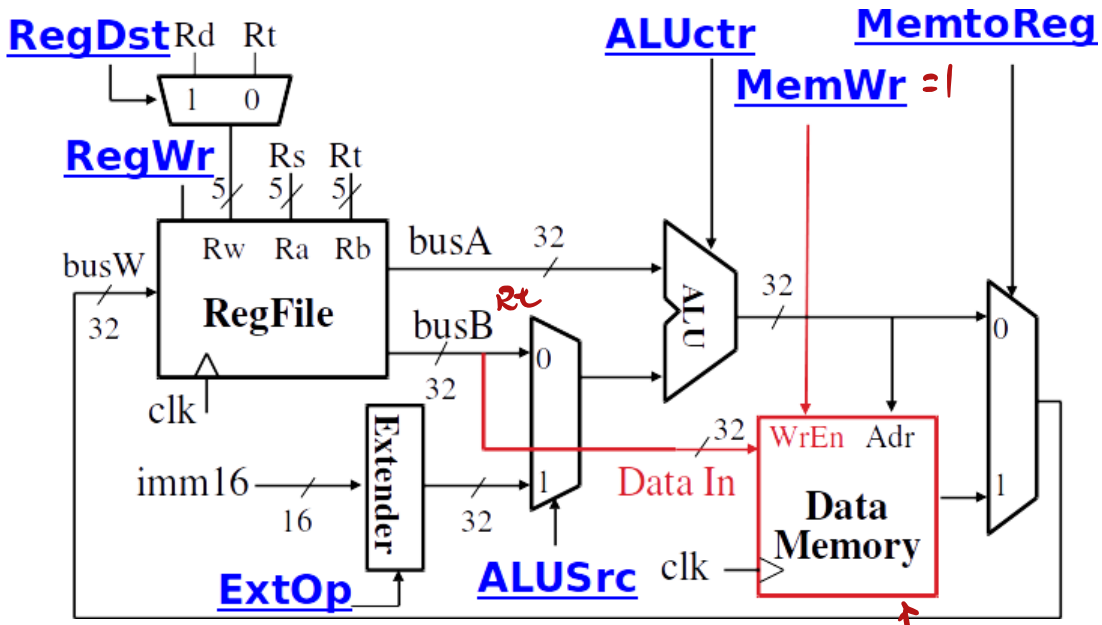
`lw $rt, off($rs)`



- Add `ExtOp` to allow for negative immediates.
- Add `MemToReg` to choose between ALU result and data read from memory.
- Arithmetic still occurs with `$rs` + `off` to get memory address.

Tracing sw

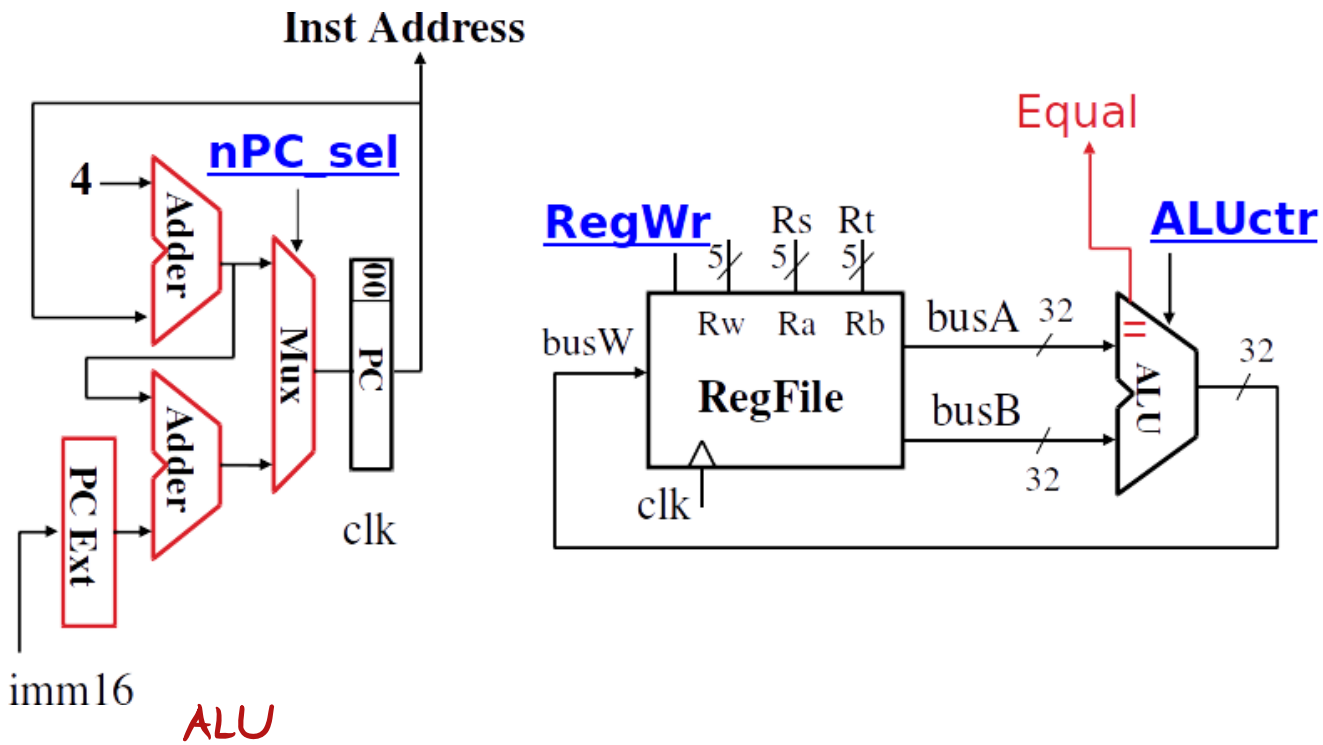
```
sw $rt, off($rs)
```



- Add a wire direct from register file to data memory.
- Add `MemWr` control to only write the read register value on a `store` instruction.
- Arithmetic still occurs with `$rs` + `off` to get memory address.

Controlling Instruction Fetch Unit: Branch instructions

`beq $rt, $rs, imm.`

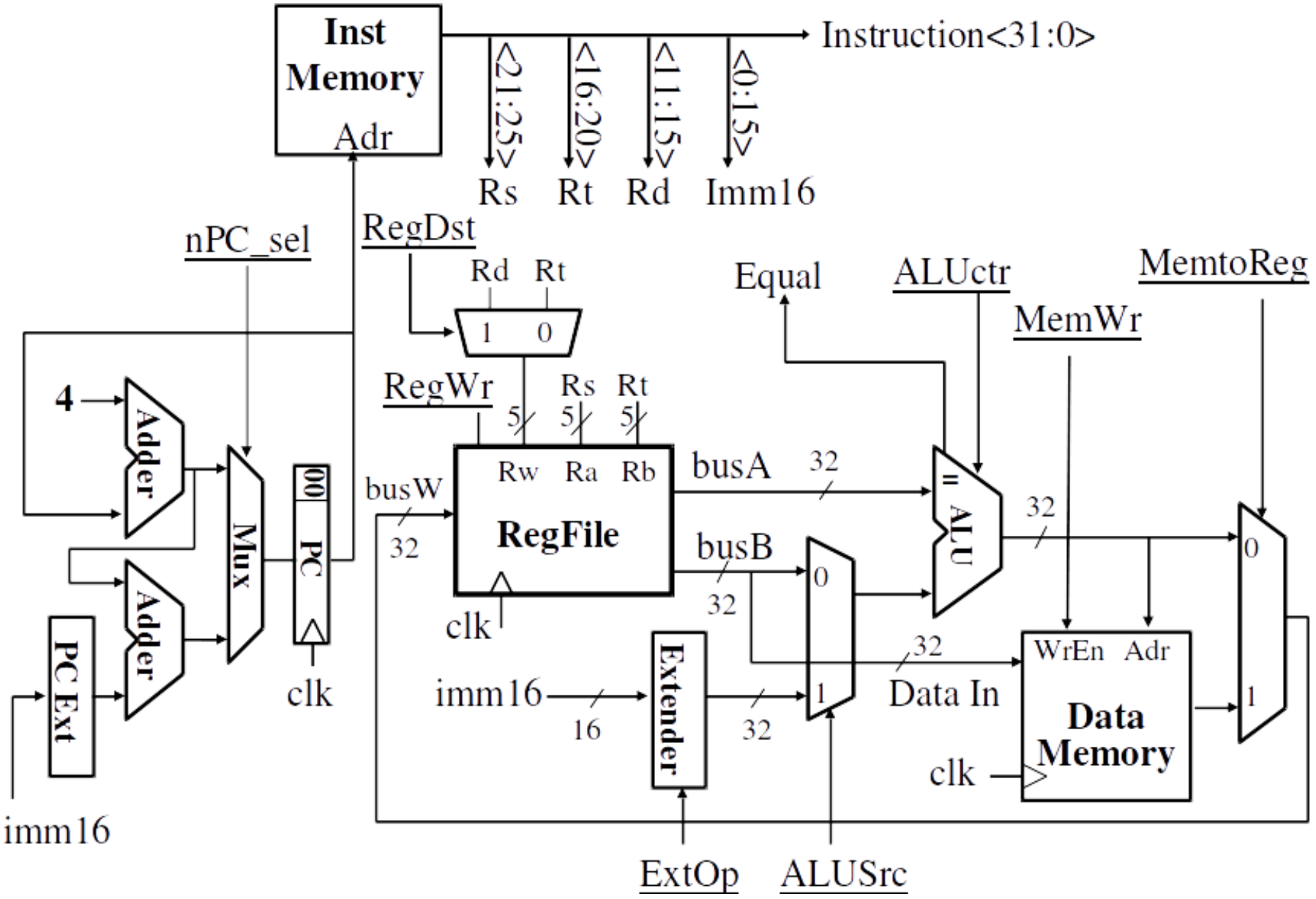


ALU

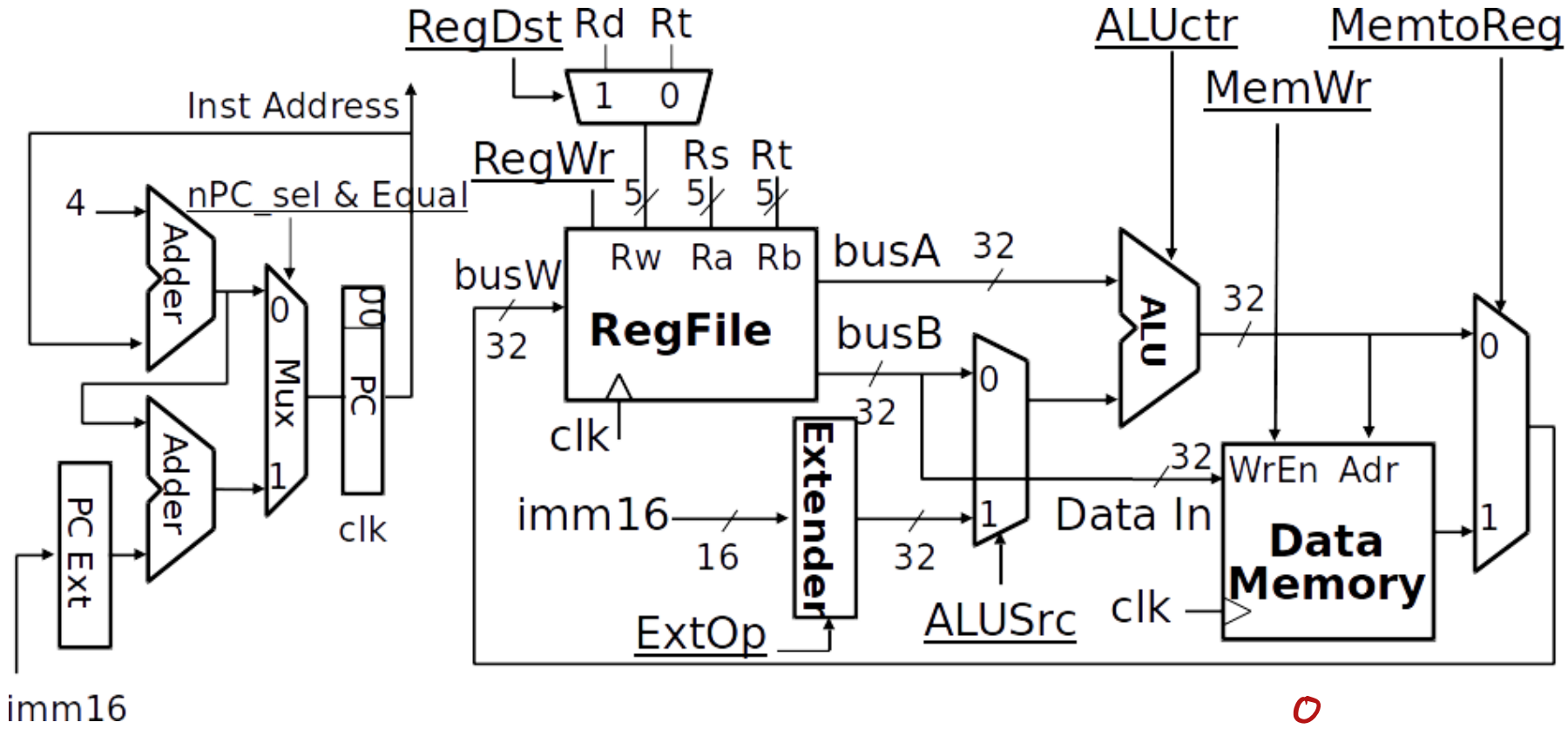
- `nPC_sel` ≡ Equal ∧ (opcode is a branch)
- If branch condition fails (if `$rs` ≠ `$rt`) or instruction is not a branch type, `PC = PC + 4`.
- Remember: datapath generally computes everything, control signals determine which results are actually read/redirected/stored/etc.

Cumulative Datapath with Control Signals

Sweet?



Control Signals Values

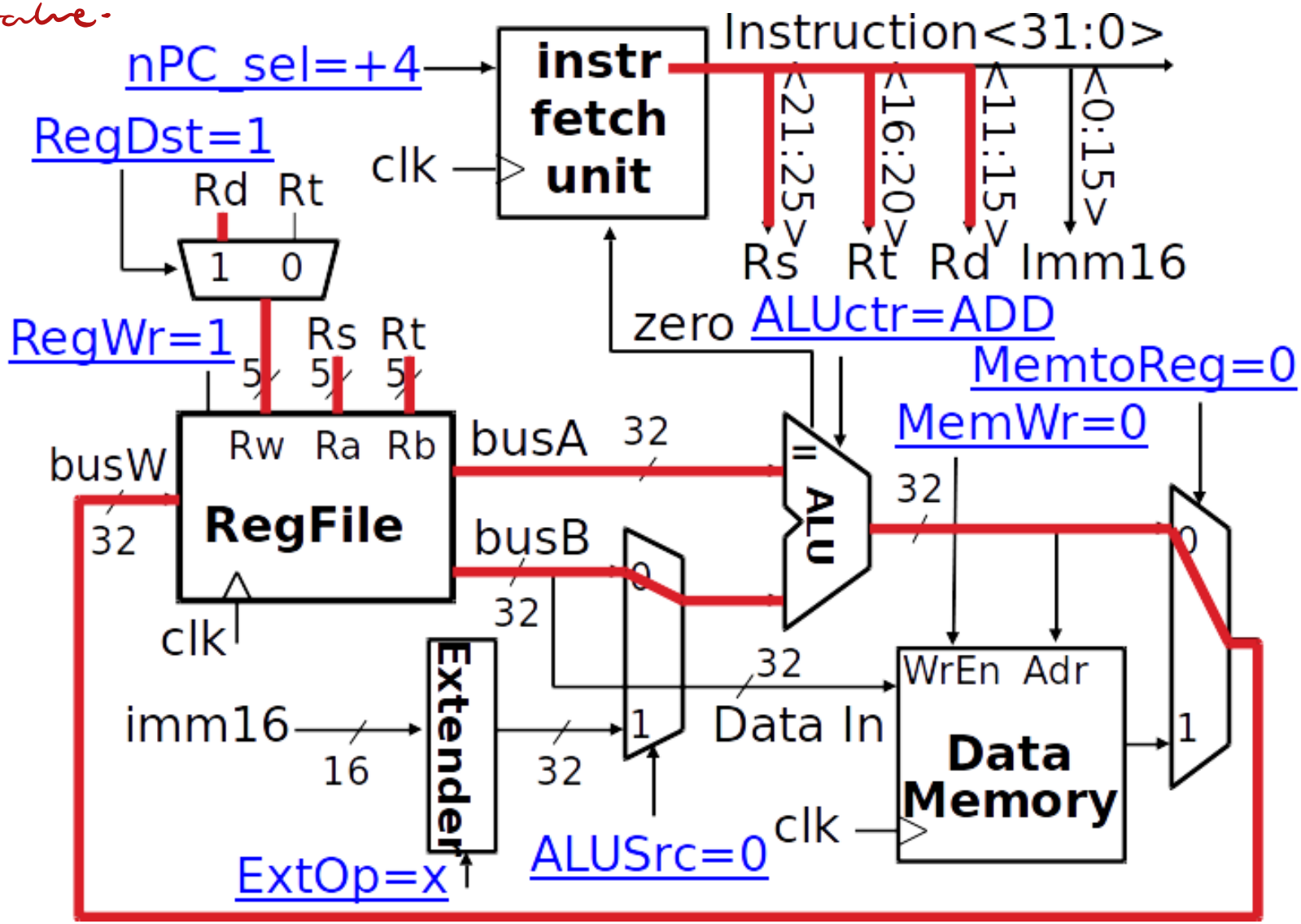


- nPC_sel: '+4', 'branch'
- RegDst: 'rd', 'rt' but you could explicit write it.
- RegWr: 1 ⇒ 'write'
- ExtOp: 'zero', 'signed'
- ALUSrc: 'rt'/'busB', imm.
- MemWr: 1 ⇒ 'write'
- MemToReg: 'ALU', 'Mem'
- ALUCtr: 'add', 'sub', '<', '>', '==', '!=', 'or', 'and', ...

Tracing add in full

highlight indicate a "meaningful" value.

```
add $rd, $rs, $rt
```



Summary of Control Signals

func	10 0000	10 0010	Doesn't Matter				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPC_sel	0	0	0	0	0	1	?
Jump	0	0	0	0	0	0	1
ExtOp	x	x	1	1	1	x	x
ALUctr	Add	Subtract	Or	Add	Add	Equal	x

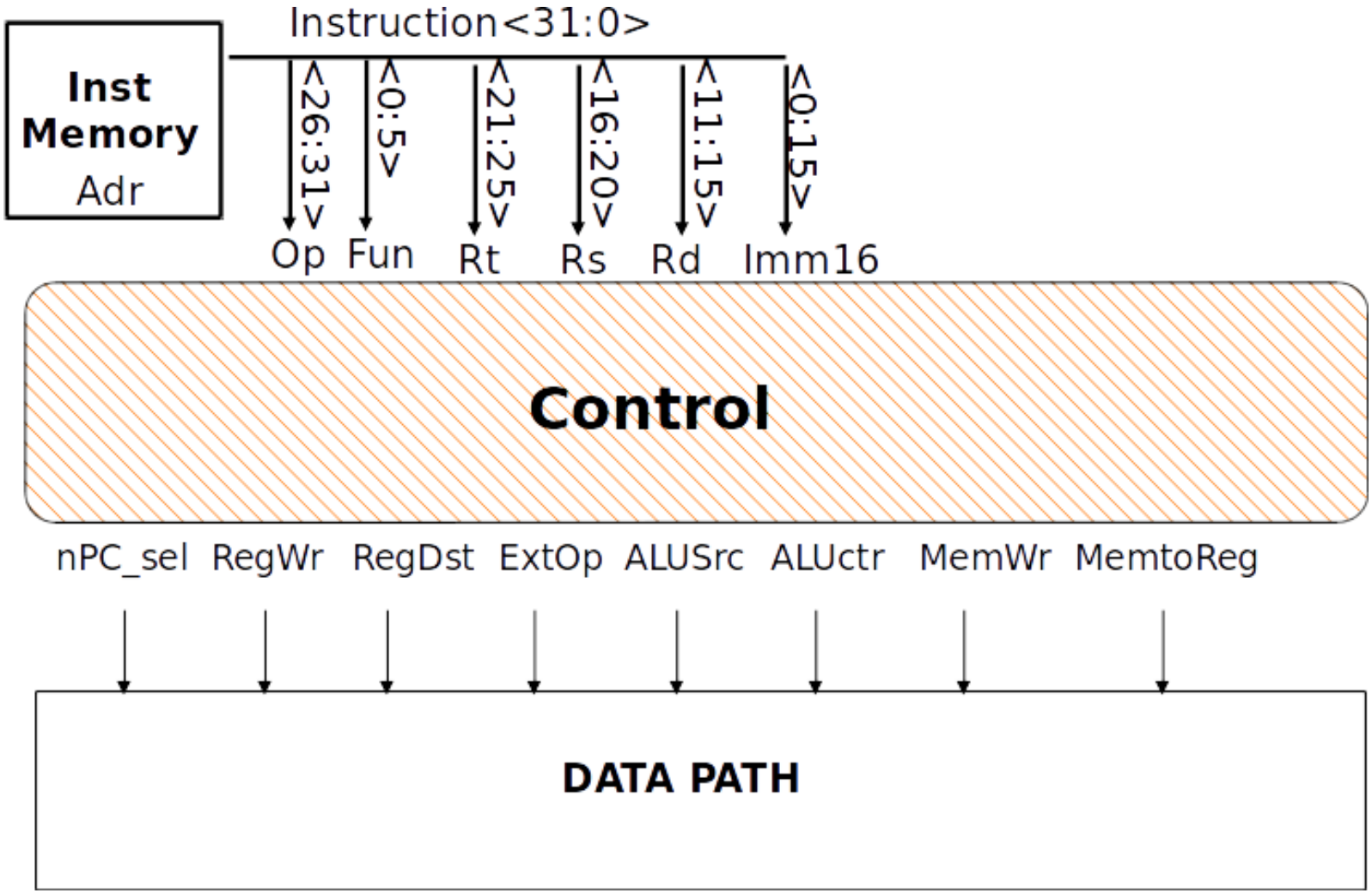
x = Don't care / Doesn't matter

Note: numeric values not really important. Just gives semantic meaning.

- e.g. `RegDst` = 'rd'
- e.g. `nPC_sel` = 'branch'

Outline

The Controller: Many Inputs, Many Ouputs



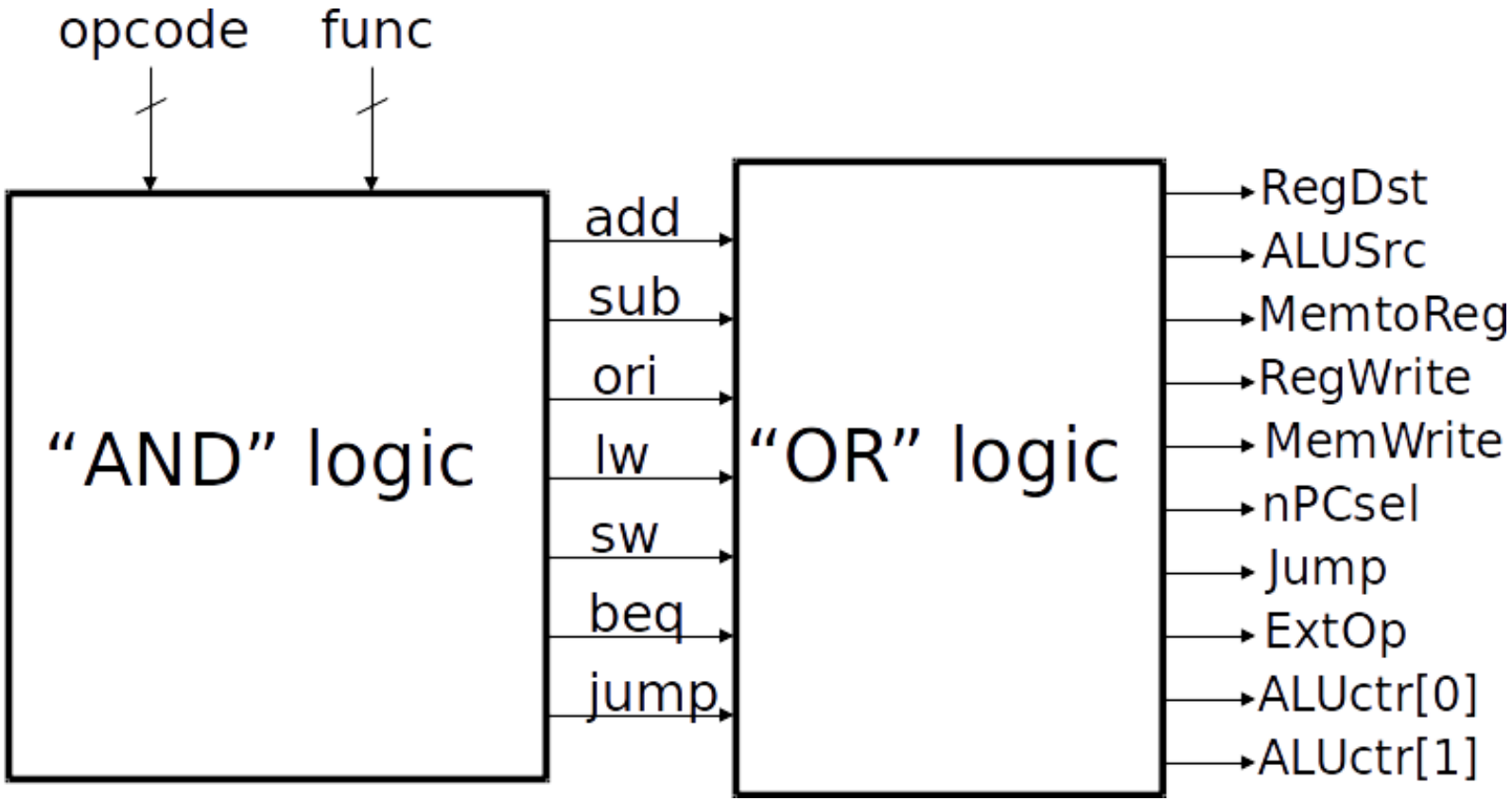
Possible Boolean Expressions for Controller

$$
\begin{aligned}
\text{RegDst} &= \text{add} + \text{sub} \\
\text{ALUSrc} &= \text{ori} + \text{lw} + \text{sw} \\
\text{MemtoReg} &= \text{lw} \\
\text{RegWrite} &= \text{add} + \text{sub} + \text{ori} + \text{lw} \\
\text{MemWrite} &= \text{sw} \\
\text{nPCsel} &= \text{beq} \\
\text{Jump} &= \text{jump} \\
\text{ExtOp} &= \text{lw} + \text{sw} \\
\text{ALUctr}[0] &= \text{sub} + \text{beq} \quad (\text{assume ALUctr is 00 ADD, 01 SUB, 10 OR}) \\
\text{ALUctr}[1] &= \text{or}
\end{aligned}
$$

Ctrl signal are boolean operation of opcode and func code.

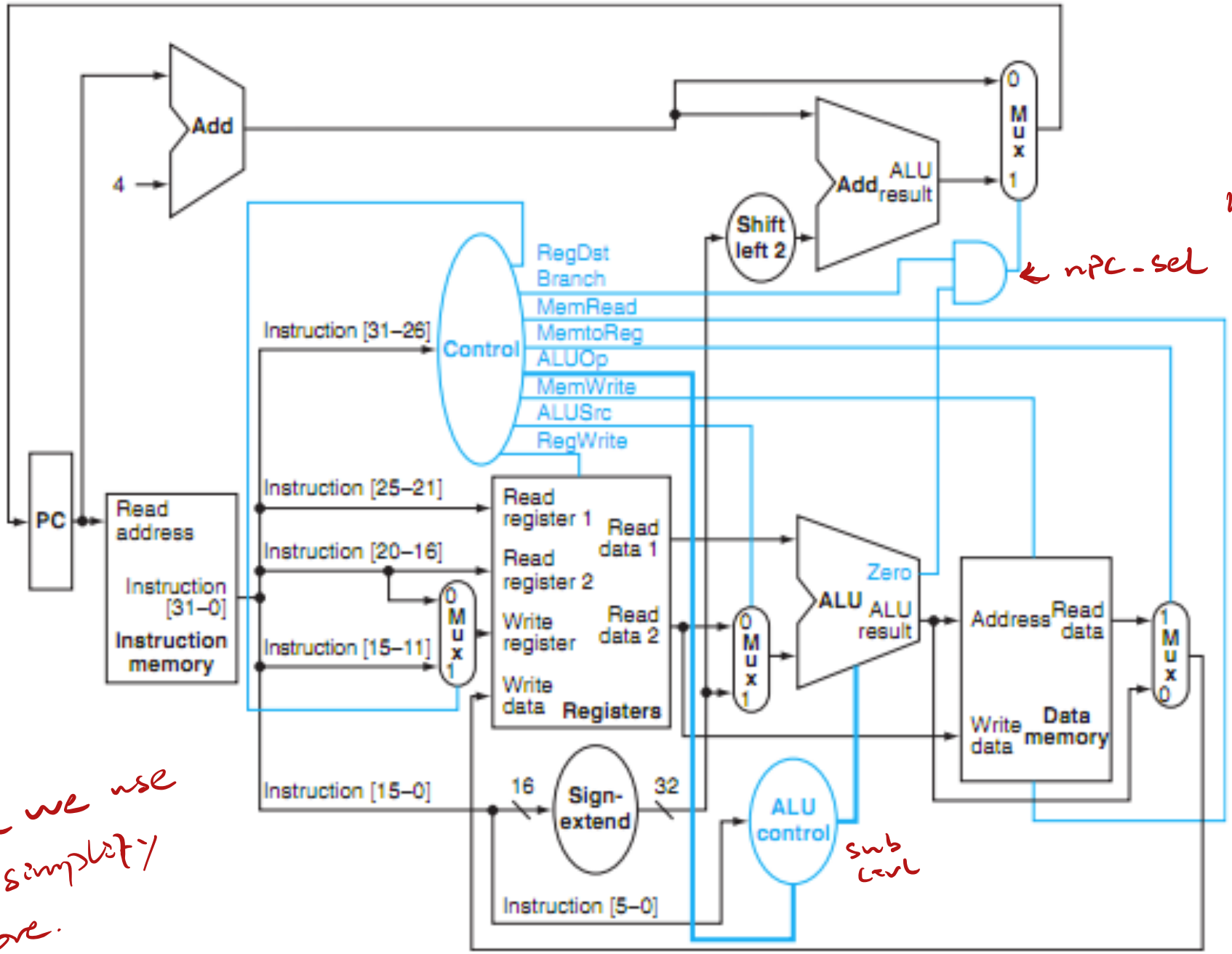
$$
\begin{aligned}
\text{rtype} &= \overline{op_5} \cdot \overline{op_4} \cdot \overline{op_3} \cdot \overline{op_2} \cdot \overline{op_1} \cdot \overline{op_0}\,, \\
\text{ori} &= \overline{op_5} \cdot \overline{op_4} \cdot \text{op}_3 \cdot \text{op}_2 \cdot \overline{op_1} \cdot \text{op}_0 \\
\text{lw} &= \text{op}_5 \cdot \overline{op_4} \cdot \overline{op_3} \cdot \overline{op_2} \cdot \text{op}_1 \cdot \text{op}_0 \\
\text{sw} &= \text{op}_5 \cdot \overline{op_4} \cdot \text{op}_3 \cdot \overline{op_2} \cdot \text{op}_1 \cdot \text{op}_0 \\
\text{beq} &= \overline{op_5} \cdot \overline{op_4} \cdot \overline{op_3} \cdot \text{op}_2 \cdot \overline{op_1} \cdot \overline{op_0} \\
\text{jump} &= \overline{op_5} \cdot \overline{op_4} \cdot \overline{op_3} \cdot \overline{op_2} \cdot \text{op}_1 \cdot \overline{op_0} \\
\text{add} &= \text{rtype} \cdot \text{func}_5 \cdot \overline{func_4} \cdot \overline{func_3} \cdot \overline{func_2} \cdot \overline{func_1} \cdot \overline{func_0} \\
\text{sub} &= \text{rtype} \cdot \text{func}_5 \cdot \overline{func_4} \cdot \overline{func_3} \cdot \overline{func_2} \cdot \text{func}_1 \cdot \overline{func_0}
\end{aligned}
$$

Implementing The Controller



Look familiar? Same scheme as a programmable logic array (PLA).

Patterson & Hennessy: Controller



next PC selection.

← nPC_sel

But we use the simplify more.

sub cool

Single Cycle Processor: Summary

- Instruction Set Architecture ↔ Datapath.
  - ↳ Instructions determine circuits needed in datapath.
  - ↳ Limitations of circuits influence allowable instructions.
- Classic RISC Datapath: `IF`, `ID`, `EX`, `MEM`, `WB`.
- Clock cycle must be long enough to account for time of *critical path* through datapath.
- MUX control flow of data through datapath.
- Controller takes opcode and `funct` as input, outputting the control signals that control MUXs, ALU, writing.
  - ↳ Boolean logic here is complex must account for every possibly combination of instructions and data.