ECE 3057: Architecture, Concurrency and Energy in Computation
Summer 2019

Lecture 3: ISA Implementation – Hardwired Non-pipelined

David V. Anderson

School of Electrical and Computer Engineering
Georgia Institute of Technology

Acknowledgment: Lecture slides adapted from MIT EECS 6.823 (Arvind and J. Emer) and ECE 3056 (Tushar Krishna).

Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

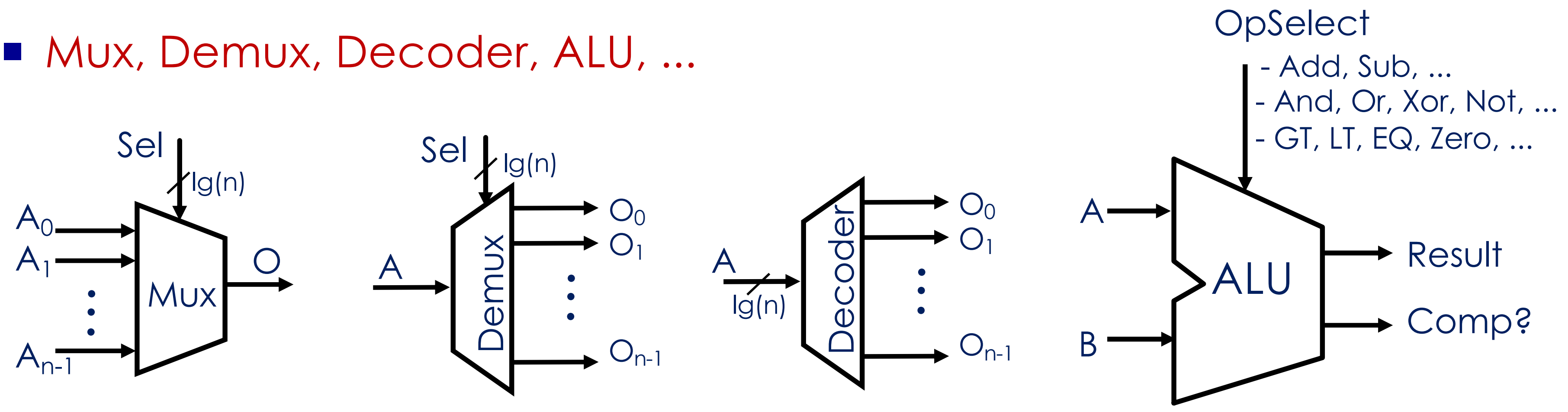
- Instructions per program depends on source code, compiler technology and ISA
- Cycles per instructions (CPI) depends upon the microarchitecture
- Time per cycle depends upon the microarchitecture and the base technology

this lecture →

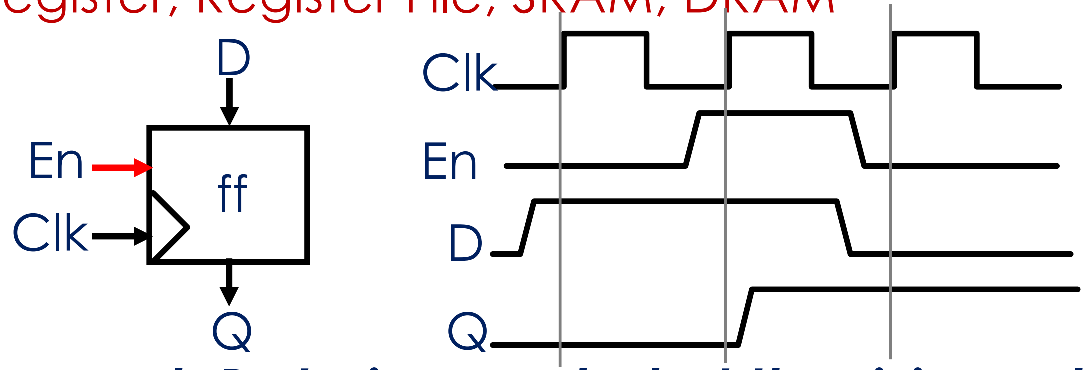
Microarchitecture	CPI	cycle time
Single-cycle unpipelined	1	long
Pipelined	1	short
Micro-coded	>1	short

Hardware Elements

- **Combinational circuits**
  - Mux, Demux, Decoder, ALU, ...

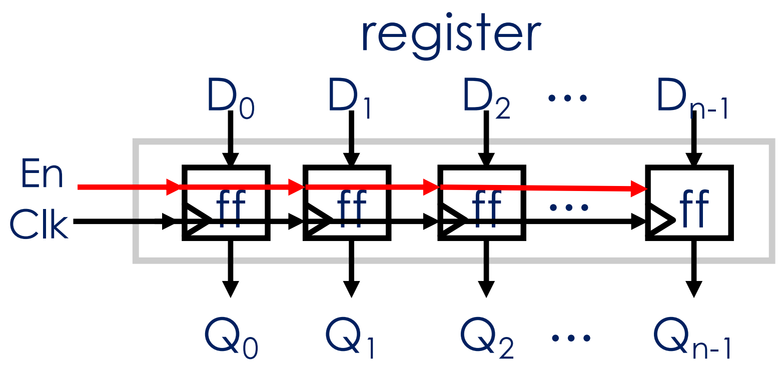


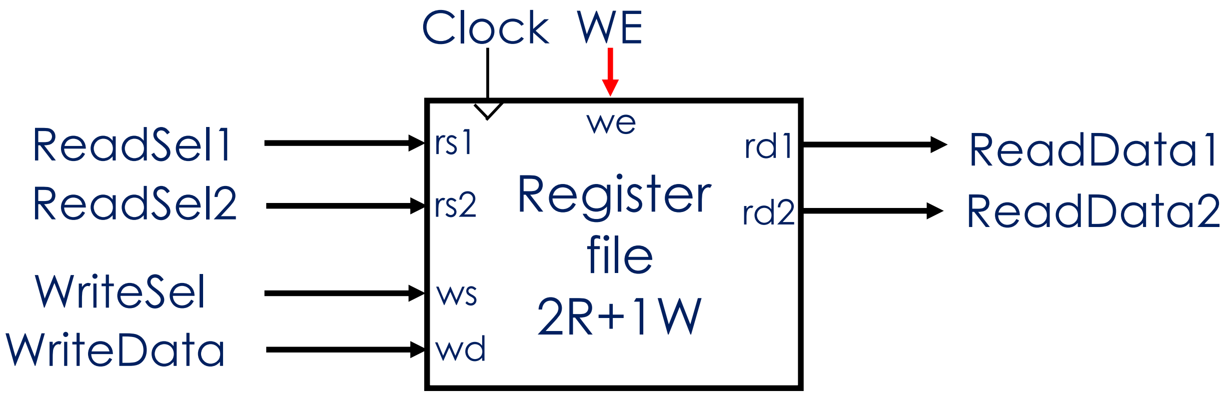
- **Synchronous state elements**
  - Flip Flop, Register, Register File, SRAM, DRAM



Edge-triggered: Data is sampled at the rising edge

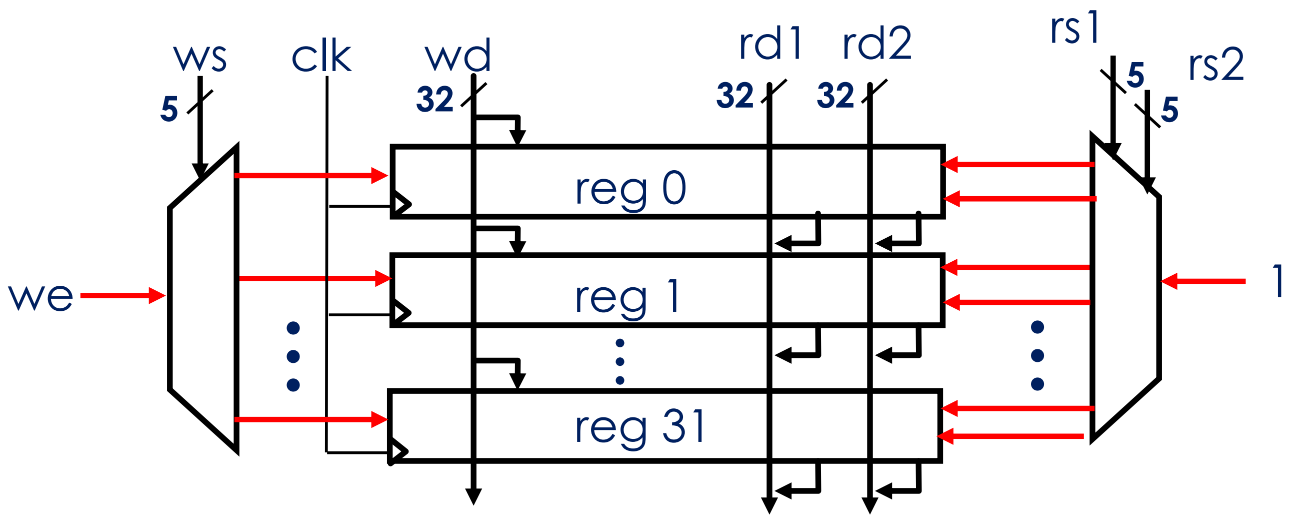
Register Files





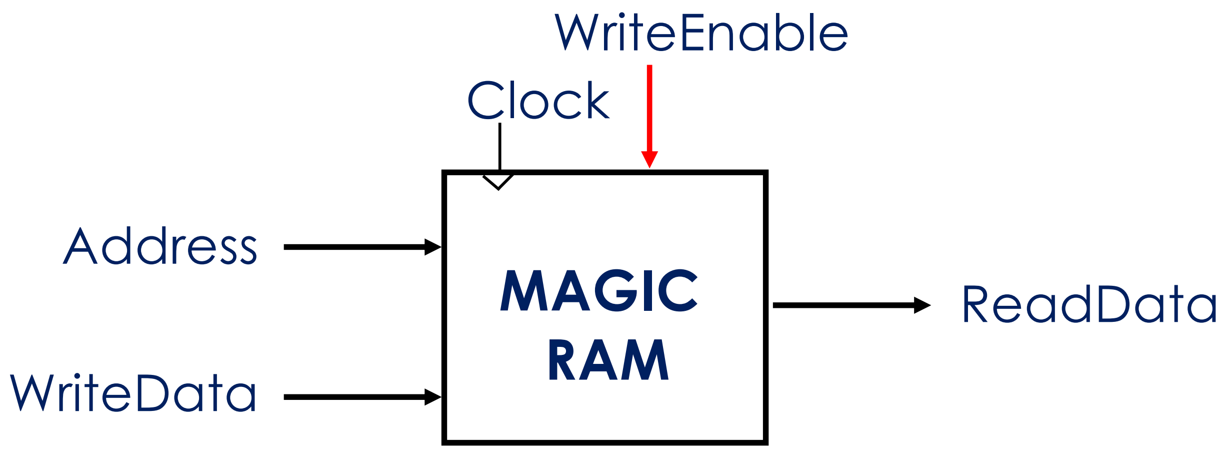
No timing issues in reading a selected register

Register File Implementation



- Register files with a large number of ports are difficult to design
  - Limits the number of reads and writes per cycle
  - Today's systems: 32-64 entry Reg File, 4-8 ports

A Simple Memory Model



- Reads and writes are always completed in one cycle
  - a Read can be done any time (i.e. combinational)
  - If enabled, a Write is performed at the rising clock edge

    (the write address and data must be stable at the clock edge)

Later in the course we will present a more realistic model of memory

Implementing MIPS:

Single-cycle per instruction
datapath & control logic

The MIPS ISA

- **Processor State**
  - 32 32-bit GPRs, R0 always contains a 0
  - 32 single precision FPRs, may also be viewed as
    - 16 double precision FPRs
  - FP status register, used for FP compares & exceptions
  - PC, the program counter
  - some other special registers

All instructions are 32 bits

- **Data types**
  - 8-bit byte, 16-bit half word
  - 32-bit word for integers
  - 32-bit word for single precision floating point
  - 64-bit word for double precision floating point

- **Load/Store style instruction set**
  - data addressing modes- immediate & indexed
  - branch addressing modes- PC relative & register indirect
  - Byte addressable memory- big endian mode

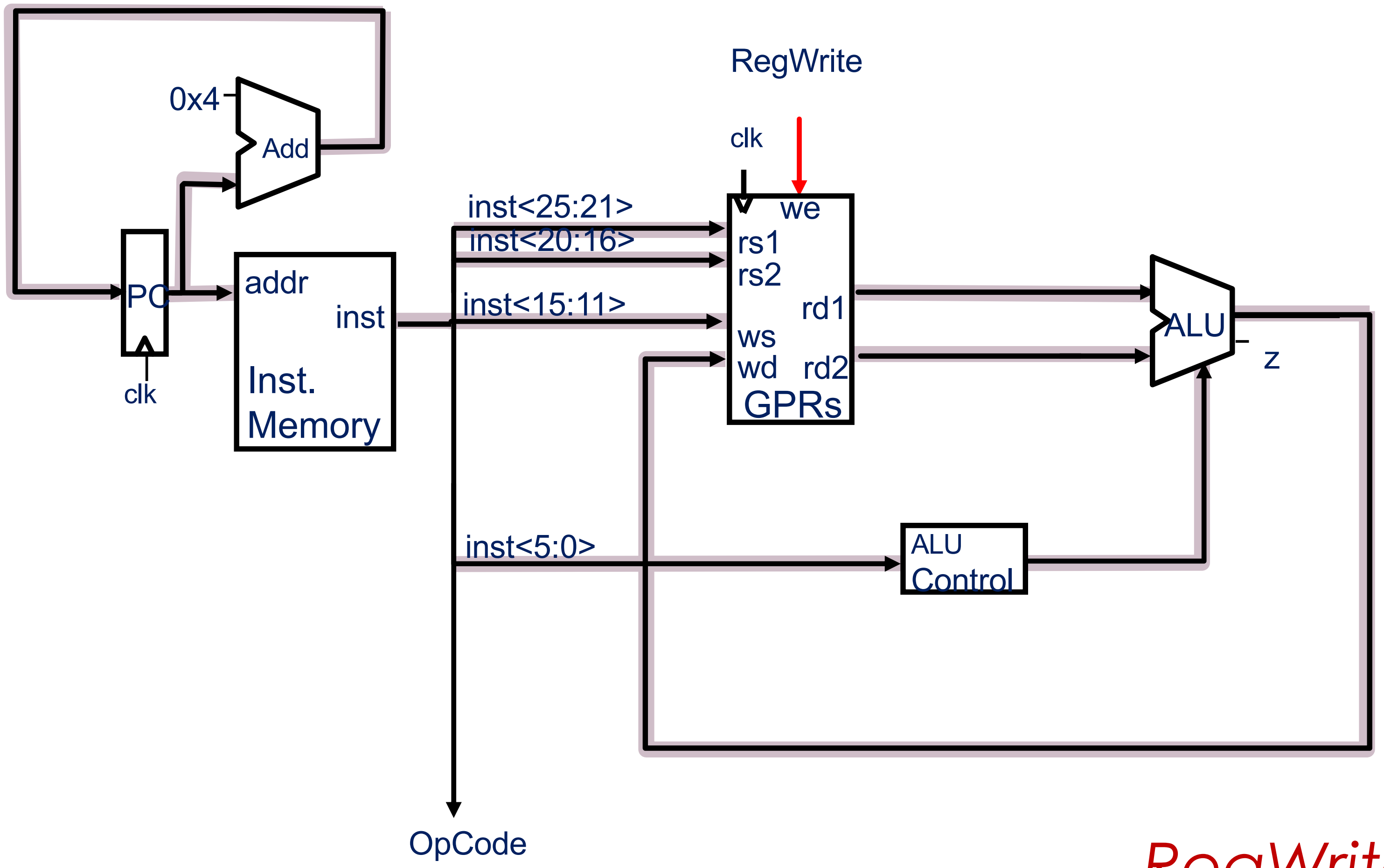
Instruction Execution

Execution of an instruction involves

1. Instruction fetch
2. Decode and Register fetch
3. ALU operation
4. Memory operation (optional)
5. Write back

and the computation of the address of the *next instruction*

Datapath: Reg-Reg ALU Instructions (R-Type)

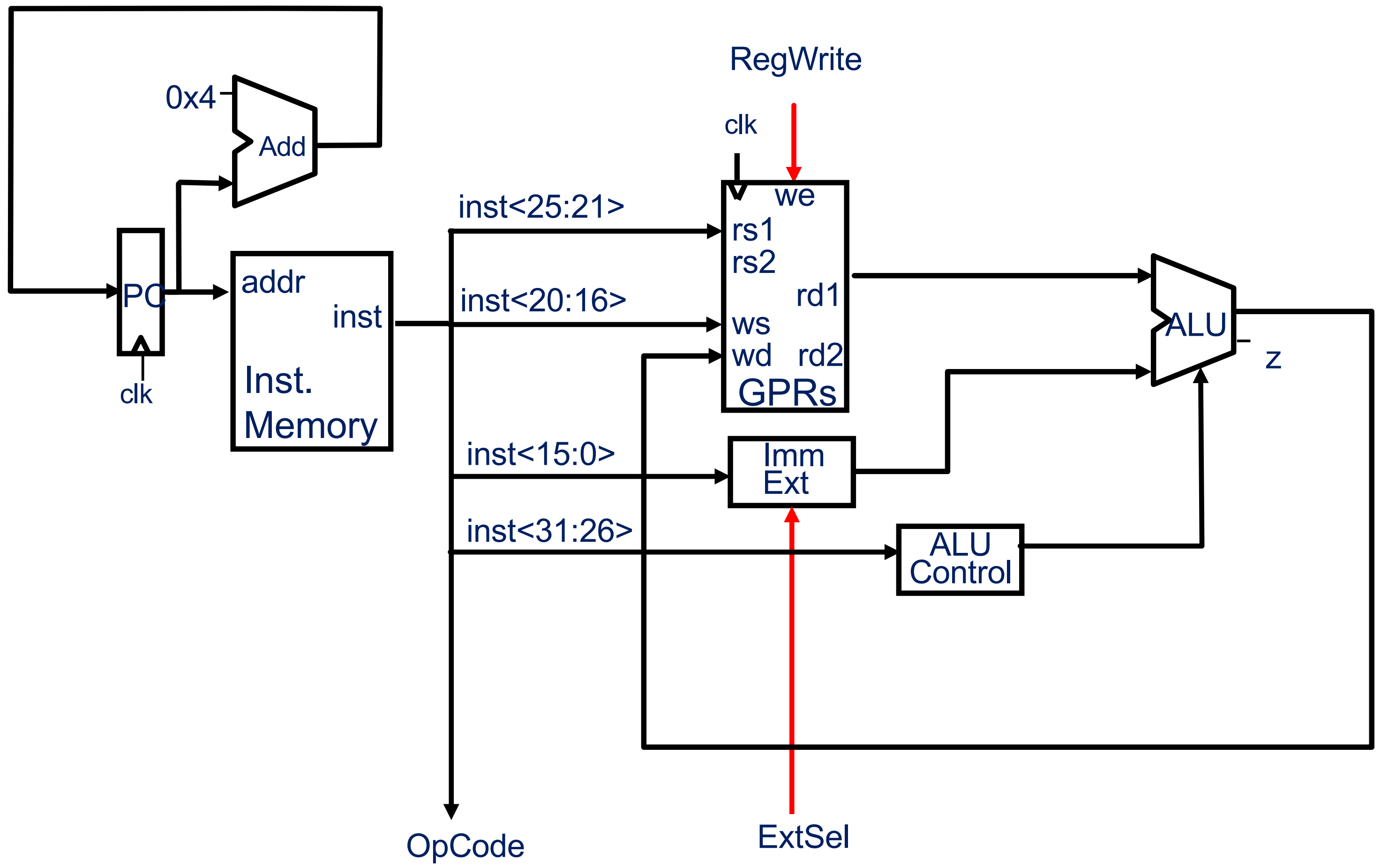


RegWrite Timing?

6	5	5	5	5	6
0	rs	rt	rd	0	func
31 26	25 21	20 16	15 11		5 0

rd ← (rs) func (rt)

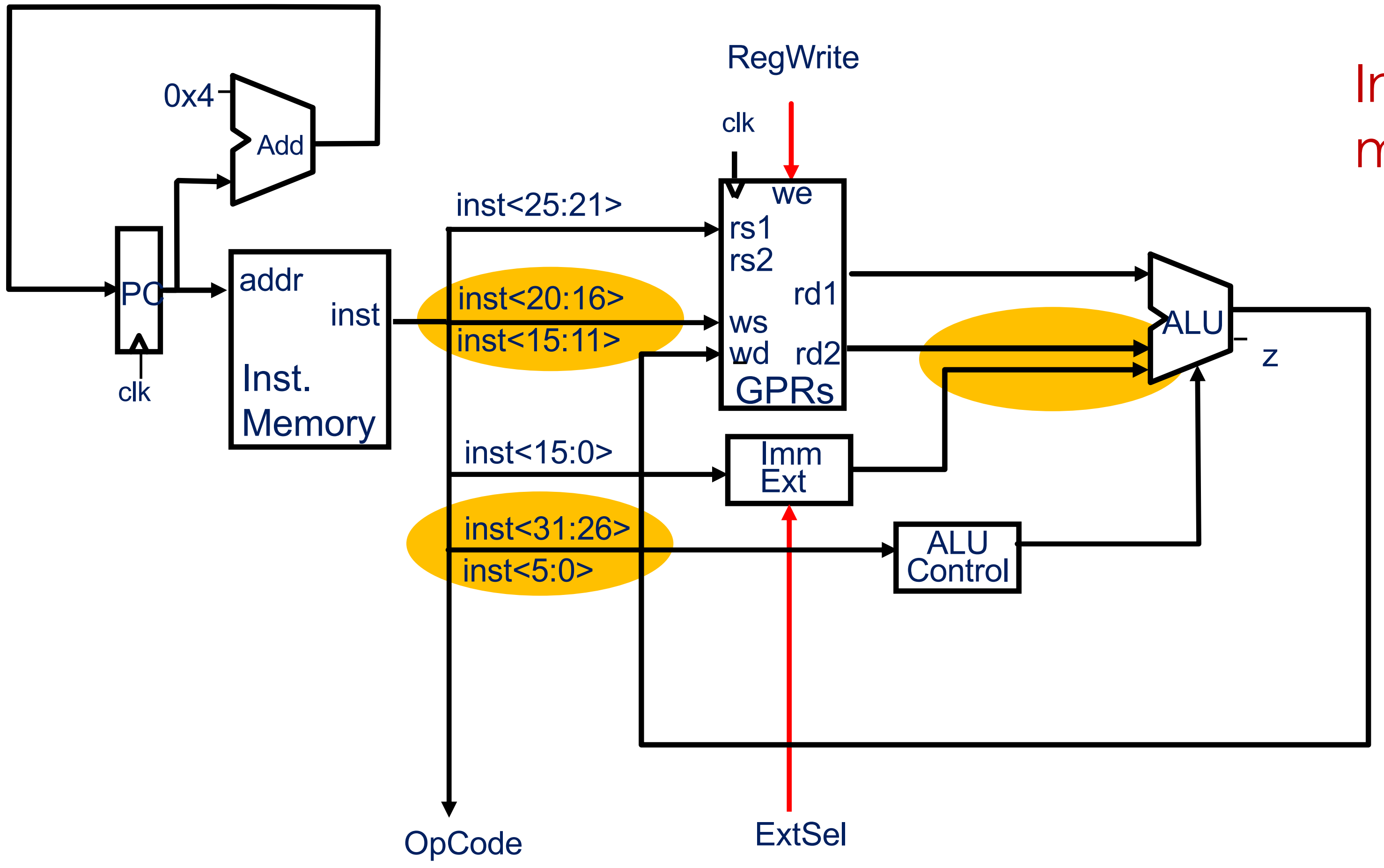
Datapath: Reg-Imm ALU Instructions (I-Type)



6	5	5	16
opcode	rs	rt	immediate
31 26	25 21	20 16	15 0

rt ← (rs) op immediate

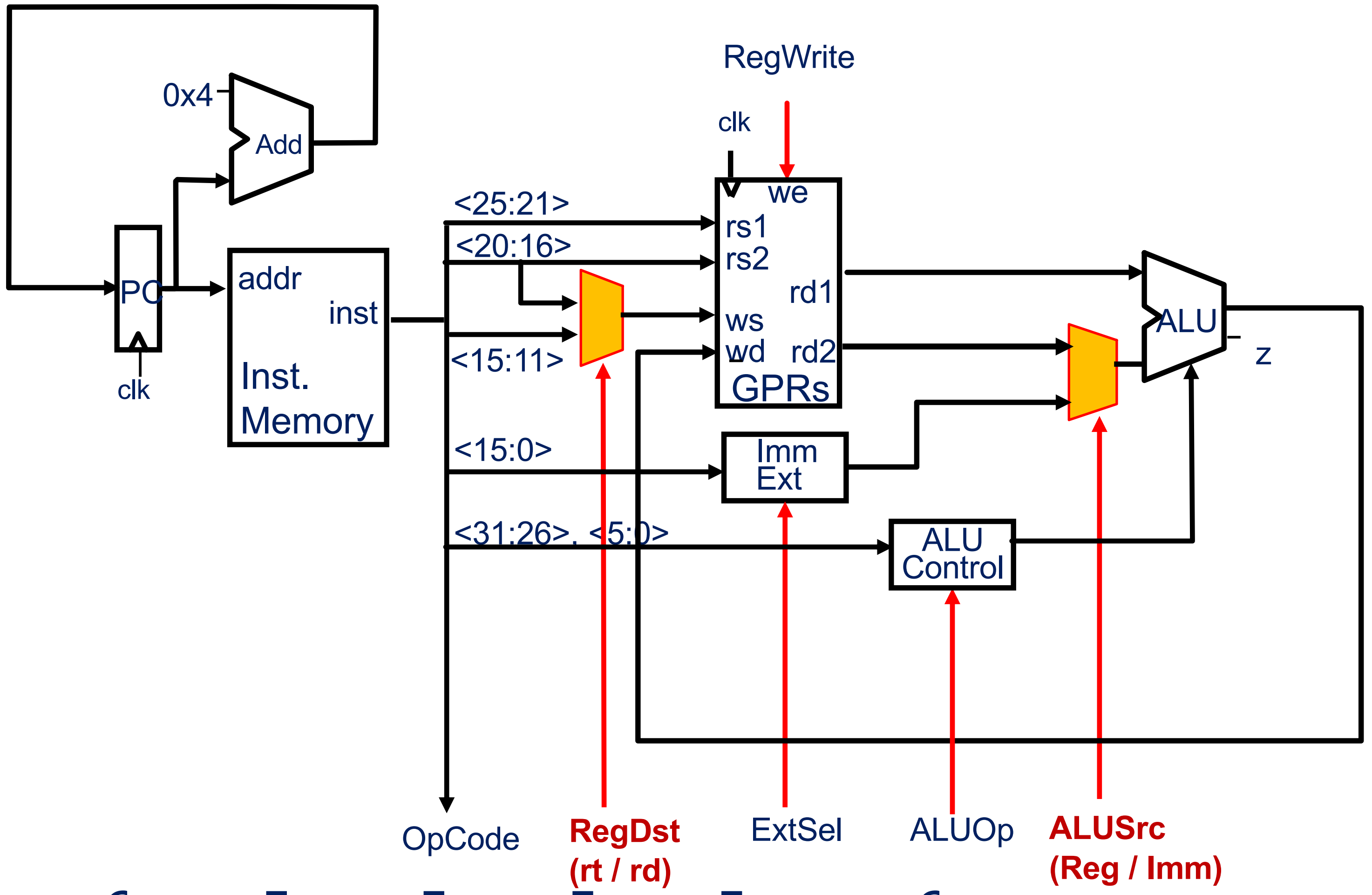
Conflicts in Merging Datapath



Introduce muxes

6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Datapath for ALU Instructions



6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Datapath for Memory Instructions

Should program and data memory be separate?

Harvard style: *separate* (Howard Aiken (Mark I) influence)

- read-only program memory
- read/write data memory

- Note:

  There must be a way to load the program memory

Princeton style: *the same* (von Neumann's influence)

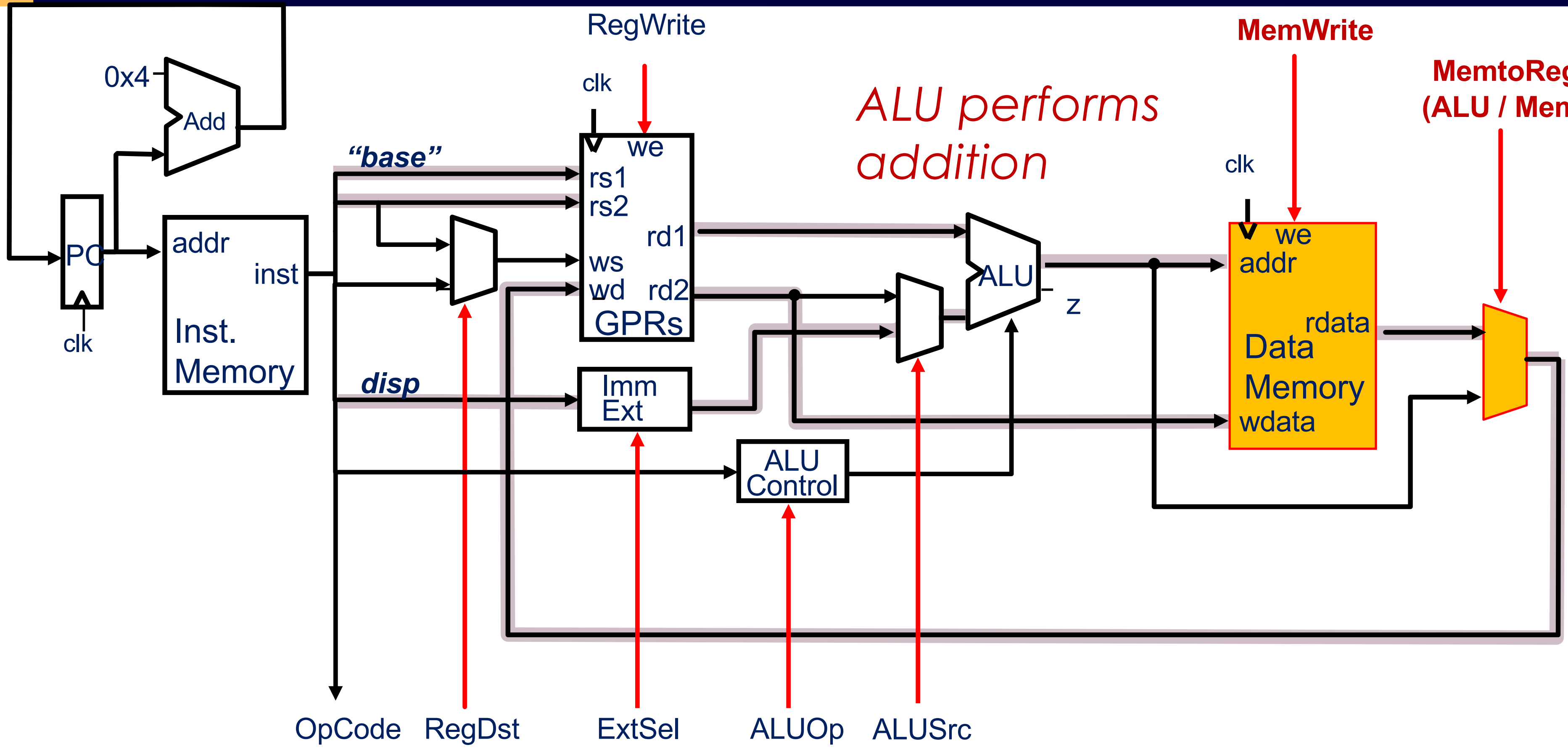
- single read/write memory for program and data

- Note:

  Executing a Load or Store instruction requires accessing the memory more than once

Harvard Datapath

Load/Store Instructions (I-Type)



6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

rs is the base register
rt is the destination of a Load or the source for a Store

MIPS Control Instructions

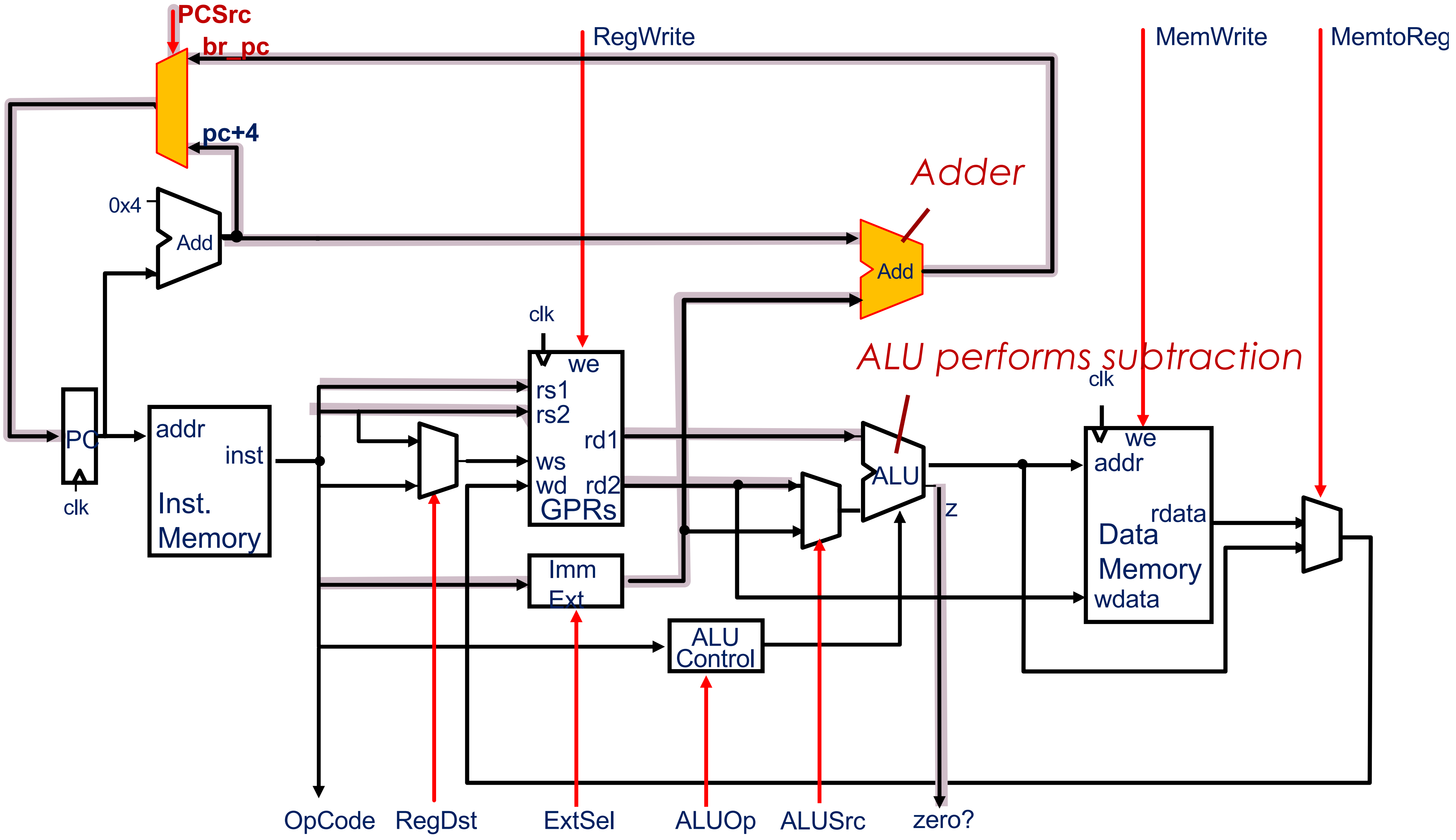
- **Conditional (on GPR) PC-relative branch**

6	5	5	16
opcode	rs	rt	offset

BEQ, BNE

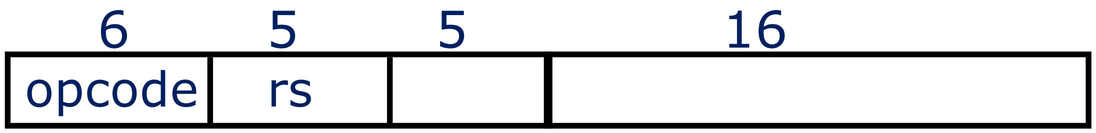
  - BEQ/BNE subtracts the value of rs and rt, and adds offset to PC+4 to calculate the target address (offset is in words)

Conditional Branches (BEQ, BNE)



MIPS Control Instructions

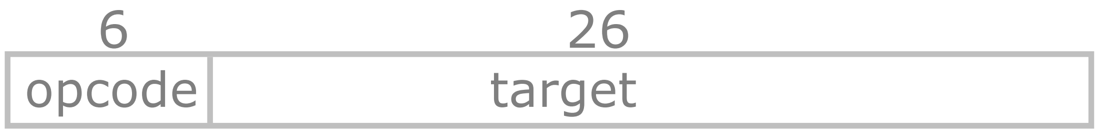
- **Unconditional register-indirect jumps**



JR, JALR

  - Jump through register (JR) jumps to address in register rs
  - Jump-&-link through register (JALR) jumps to address in register rs and stores PC+4 into the link register (R31)

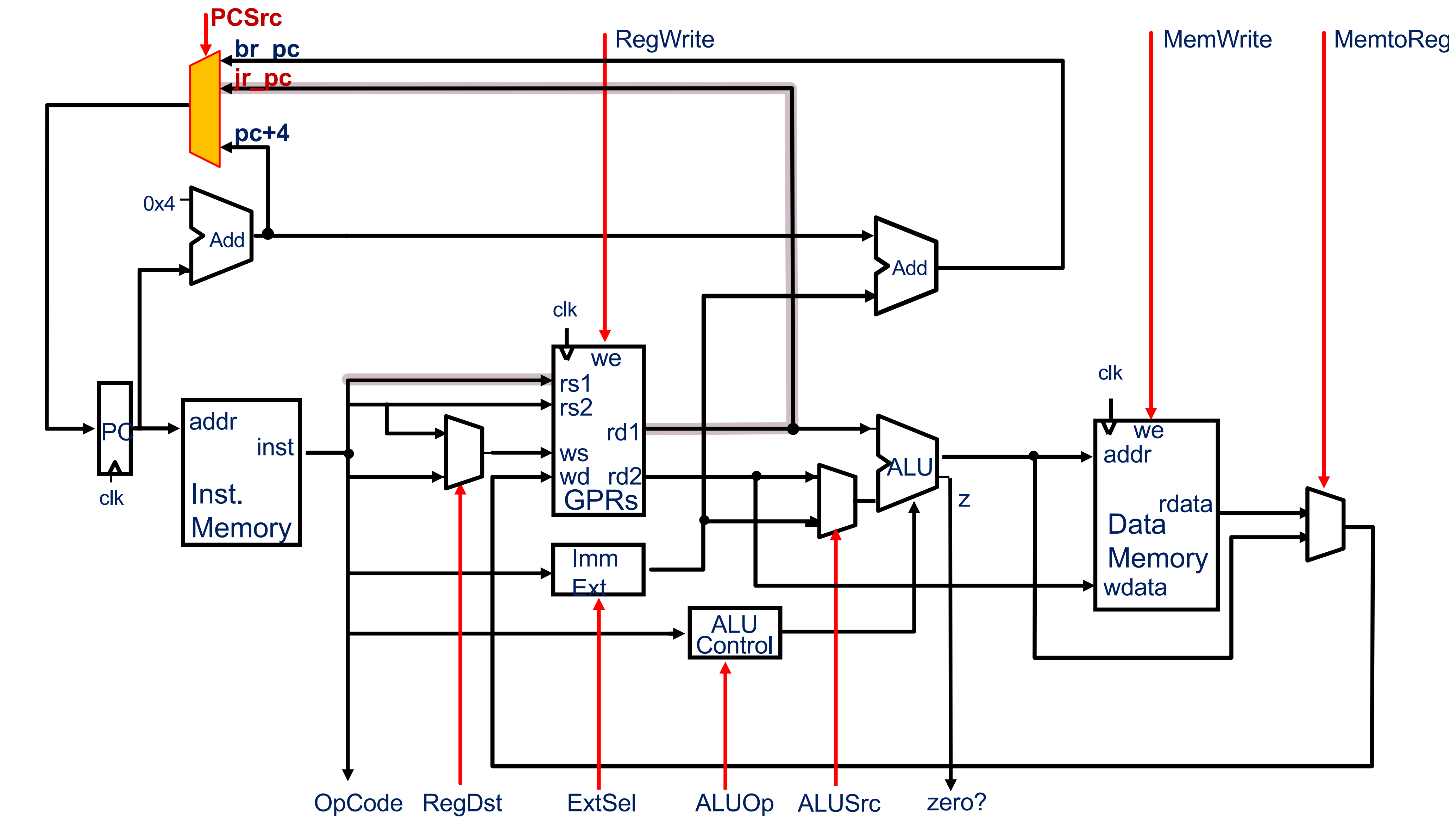
- **Unconditional absolute jumps**



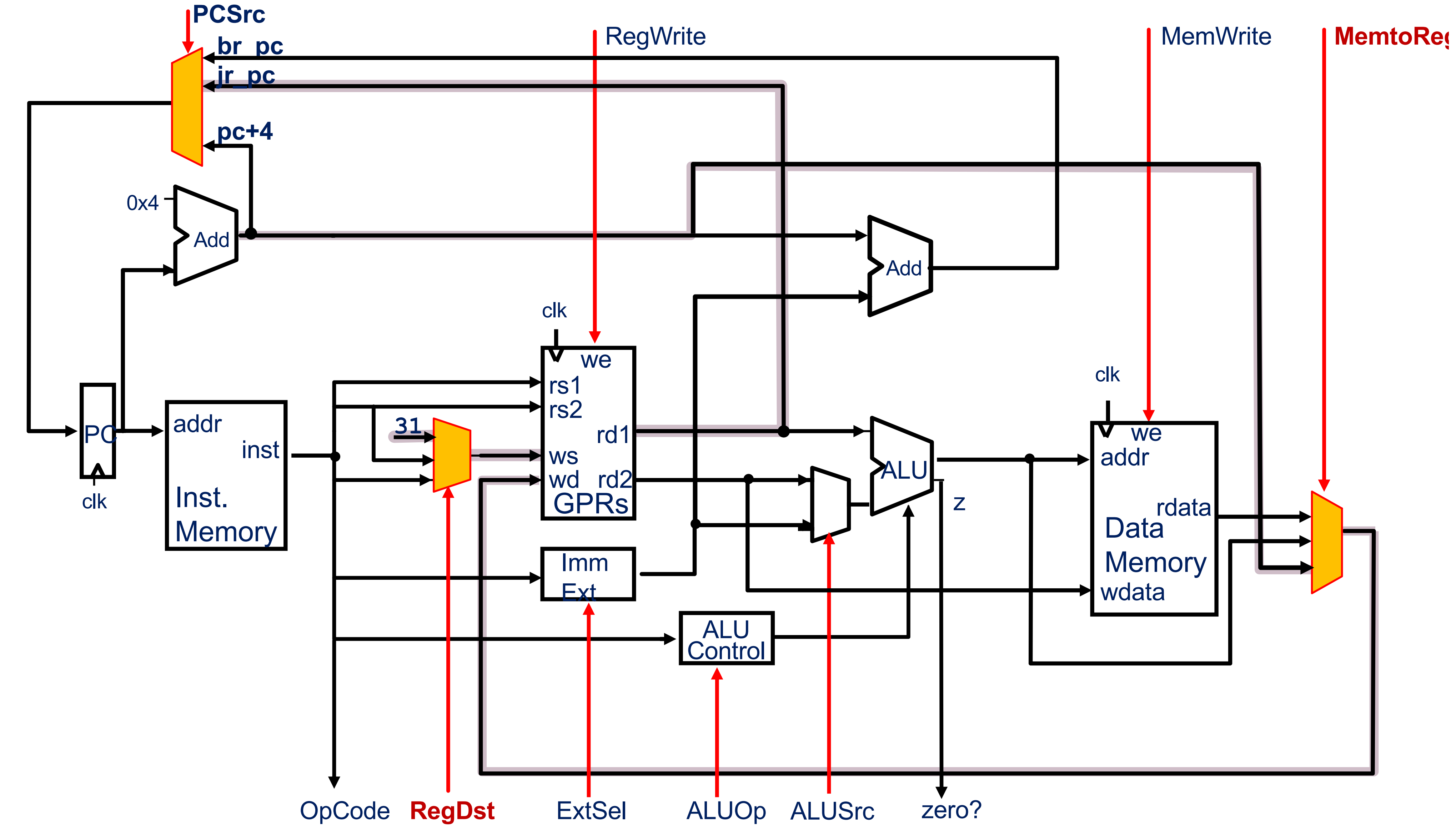
J, JAL

  - Absolute jumps append target to PC<31:28> to calculate the target address

Register-Indirect Jumps (JR)

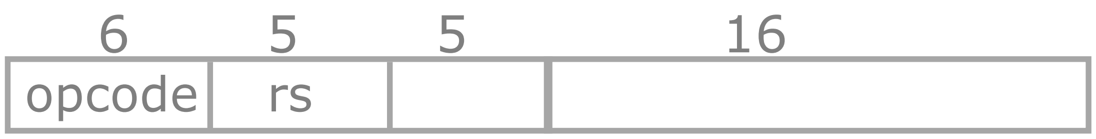


Register-Indirect Jump-&-Link (JALR)



MIPS Control Instructions

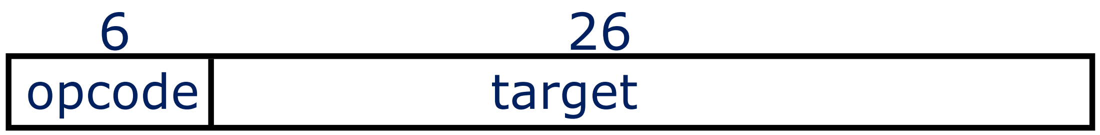
- **Unconditional register-indirect jumps**



JR, JALR

  - Jump through register (JR) jumps to address in register rs
  - Jump-&-link through register (JALR) jumps to address in register rs and stores PC+4 into the link register (R31)

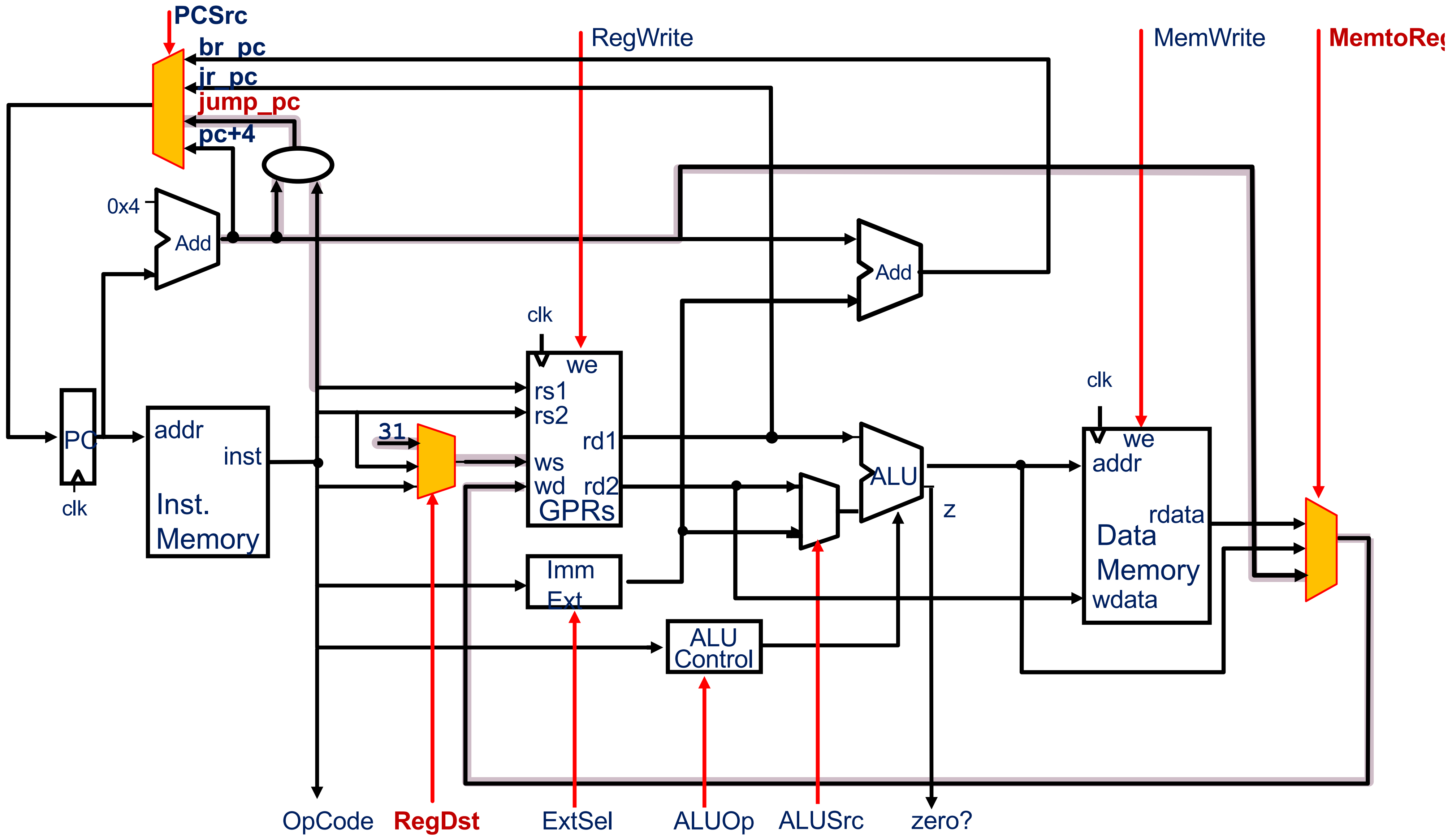
- **Unconditional absolute jumps**



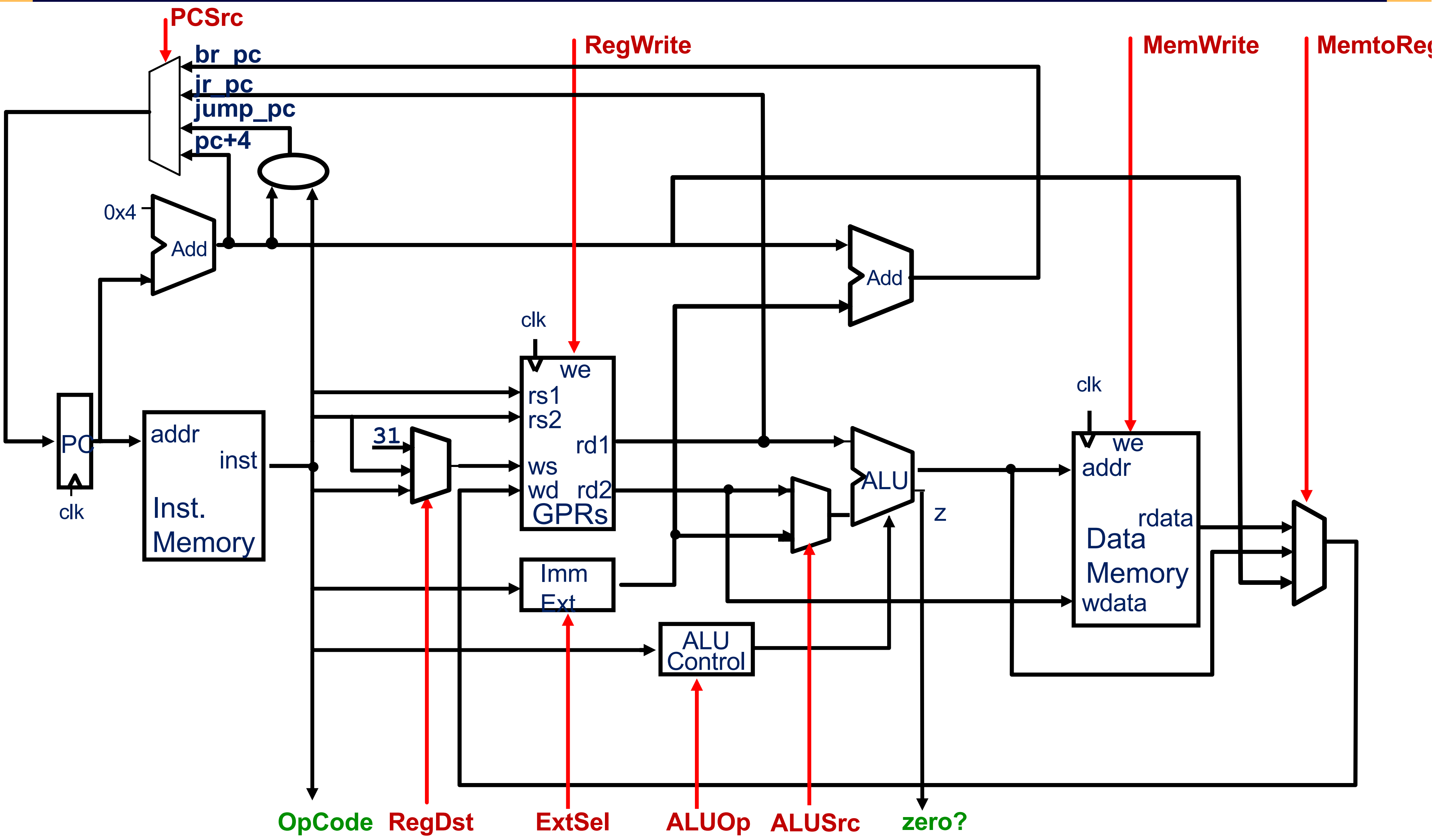
J, JAL

  - Absolute jumps append target to PC<31:28> to calculate the target address

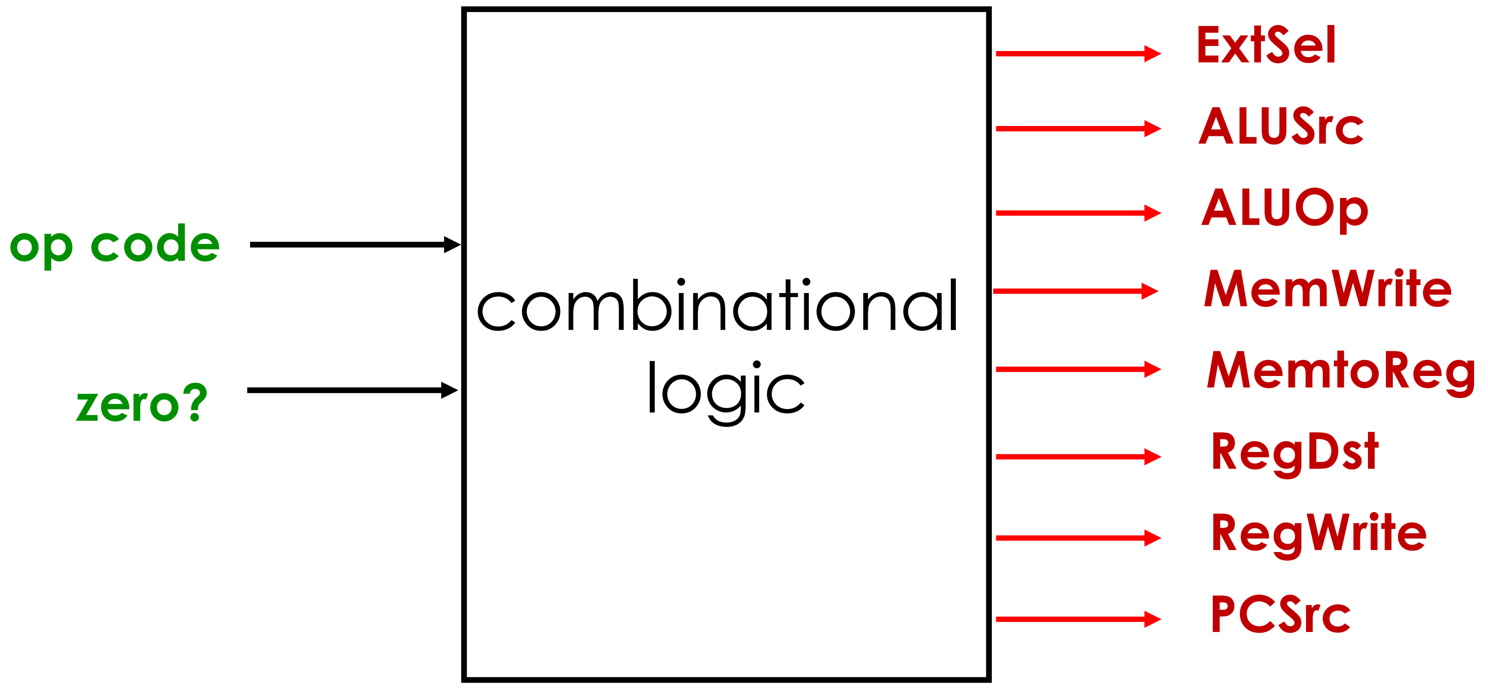
Absolute Jumps (J, JAL)



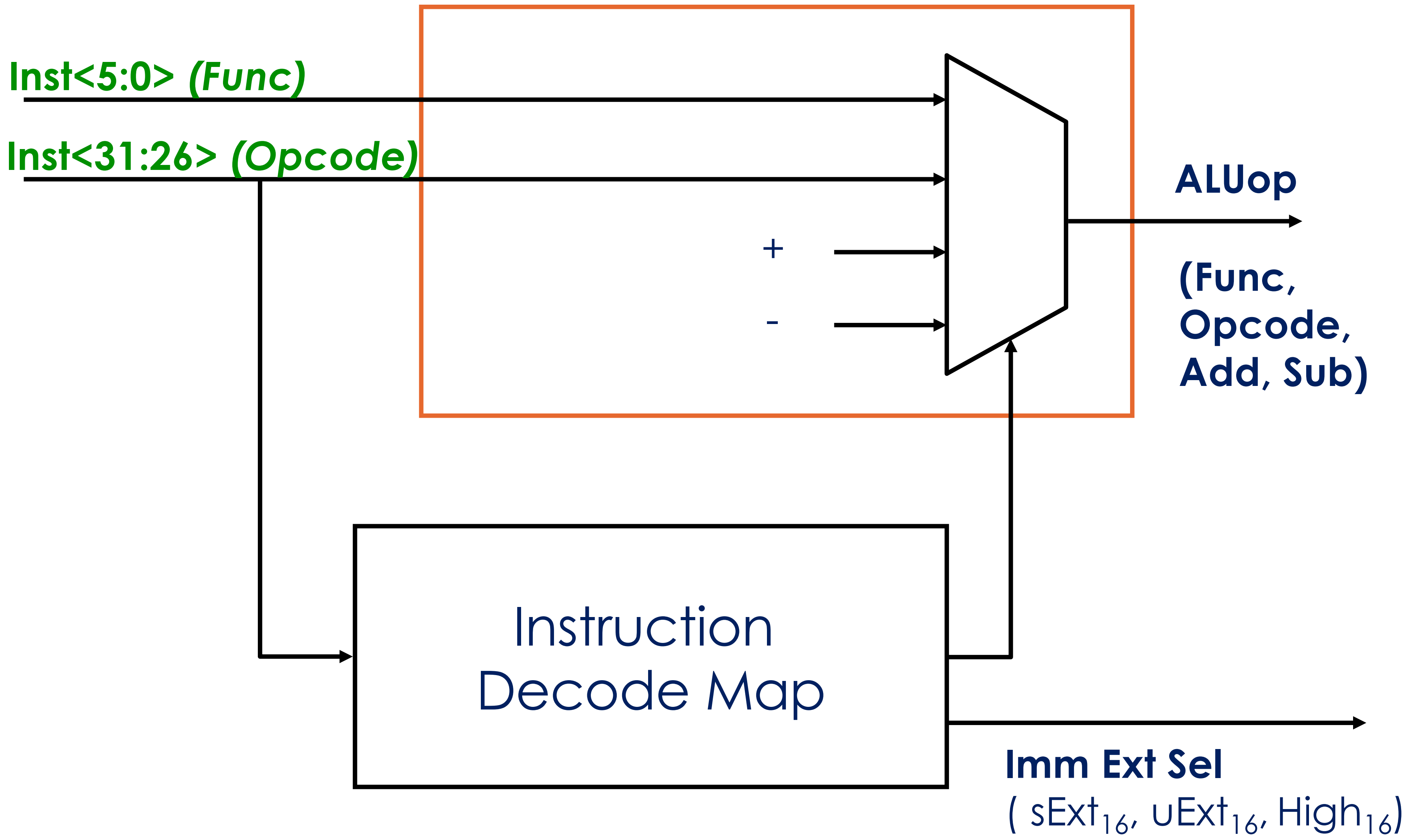
Summary: Harvard-Style Datapath for MIPS



Hardwired Control is pure Combinational Logic



ALU Control & Immediate Extension



Hardwired Control Table

Opcode	ExtSel	ALUSrc	ALUOp	MemW	RegW	MemtoReg	RegDst	PCSrc
ALU	*	Reg	Func	no	yes	ALU	rd	pc+4
ALUi	$sExt_{16}$	Imm	Op	no	yes	ALU	rt	pc+4
ALUiu	$uExt_{16}$	Imm	Op	no	yes	ALU	rt	pc+4
LW	$sExt_{16}$	Imm	+	no	yes	Mem	rt	pc+4
SW	$sExt_{16}$	Imm	+	yes	no	*	*	pc+4
$BEQ_{z=0}$	$sExt_{16}$	Reg	-	no	no	*	*	br_pc
$BEQ_{z=1}$	$sExt_{16}$	Reg	-	no	no	*	*	pc+4
J	*	*	*	no	no	*	*	jump_pc
JAL	*	*	*	no	yes	PC	R31	jump_pc
JR	*	*	*	no	no	*	*	jr_pc
JALR	*	*	*	no	yes	PC	R31	jr_pc

ExtSel = $sExt_{16}$, $uExt_{16}$ **ALUSrc** = Reg / Imm **ALUOp** = Func/Op/+/-
MemW: N/Y **RegW**: N/Y **MemtoReg** = ALU / Mem / PC
RegDst = rt / rd / R31 **PCSrc** = pc+4 / br_pc / jump_pc / jr_pc

Single-Cycle Hardwired Control:
Harvard architecture

We will assume

- clock period is sufficiently long for all of the following steps to be “completed”:

  1. instruction fetch
  2. decode and register fetch
  3. ALU operation
  4. data fetch if required
  5. register write-back setup time

$$\Rightarrow\ t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB}$$

- At the rising edge of the following clock, the PC, the register file and the memory are updated