

# Instruction Set Architecture & Hardwired, Non-pipelined ISA Implementation

*Daniel Sanchez*

Computer Science & Artificial Intelligence Lab  
M.I.T.



# Programmer's view of a machine: IBM 650

---

A drum machine with 44 instructions

Instruction: 60 1234 1009

- “Load the contents of location 1234 into the *distribution*; put it also into the *upper accumulator*; set *lower accumulator* to zero; and then go to location 1009 for the next instruction.”

- Programmer's view of the machine was inseparable from the actual hardware implementation
- Good programmers optimized the placement of instructions on the drum to reduce latency!

# Compatibility Problem at IBM

---

By early 60's, *IBM had 4 incompatible lines of computers!*

701	→	7094
650	→	7074
702	→	7080
1401	→	7010

Each system had its own

- Instruction set
- I/O system and Secondary Storage:  
magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche  
business, scientific, real time, ...

⇒ *IBM 360*

# IBM 360: Design Premises

*Amdahl, Blaauw, and Brooks, 1964*

---

The design must lend itself to *growth and successor machines*

- General method for connecting I/O devices
- Total performance - answers per month rather than bits per microsecond  $\Rightarrow$  *programming aids*
- Machine must be capable of *supervising itself* without manual intervention
- Built-in *hardware fault checking* and locating aids to reduce down time
- Simple to assemble systems with redundant I/O devices, memories, etc. for *fault tolerance*
- Some problems required floating point words larger than 36 bits

# Processor State and Data Types

---

*The information held in the processor at the end of an instruction to provide the processing context for the next instruction.*

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.
- If the processing of an instruction can be interrupted then the *hardware* must save and restore the state in a transparent manner

*Programmer's machine model* is a **contract** between the hardware and software

# Instruction Set

---

*The control for changing the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.*

Some things an ISA must specify:

- *A way to reference registers and memory*
- *The computational operations available*
- *How to control the sequence of instructions*
- *A binary representation for all of the above*

*ISA must satisfy the needs of the software:  
- assembler, compiler, OS, VM*

# IBM 360: *A General-Purpose Register (GPR) Machine*

---

- Processor State
  - 16 General-Purpose 32-bit Registers
  - 4 Floating Point 64-bit Registers
  - A Program Status Word (PSW)
    - *PC, Condition codes, Control flags*
- Data Formats
  - 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double-words
  - 24-bit addresses
- A 32-bit machine with 24-bit addresses
  - *No instruction contains a 24-bit address!*
- Precise interrupts



# IBM 360: Initial Implementations (1964)

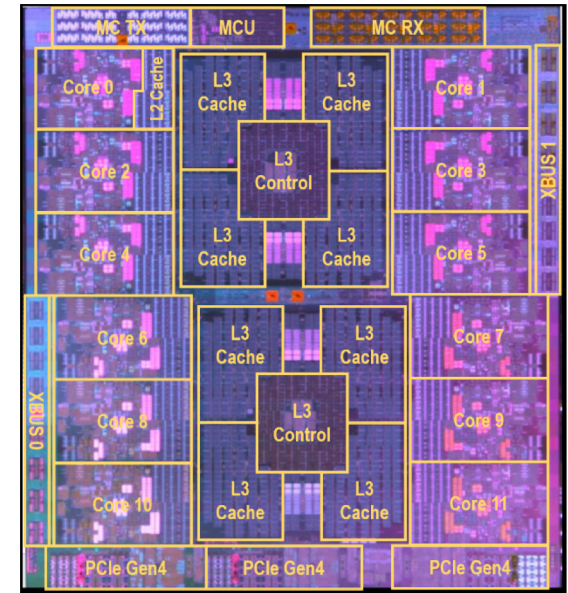
	<i>Model 30</i>	<i>. . .</i>	<i>Model 70</i>
<i>Memory Capacity</i>	8K - 64 KB		256K - 512 KB
<i>Memory Cycle</i>	2.0 $\mu$ s	<i>...</i>	1.0 $\mu$ s
<i>Datapath</i>	8-bit		64-bit
<i>Circuit Delay</i>	30 nsec/level		5 nsec/level
<i>Registers</i>	in Main Store		in Transistor
<i>Control Store</i>	Read only 1 $\mu$ sec		Dedicated circuits

- Six implementations (Models, 30, 40, 50, 60, 62, 70)
- 50x performance difference across models
- *ISA completely hid the underlying technological differences between various models*

With minor modifications, IBM 360 ISA is still in use

# IBM 360: Fifty-five years later... z15 Microprocessor

- 9.2 billion transistors, 12-core design
- Up to 190 cores (2 spare) per system
- 5.2 GHz, 14nm CMOS technology
- 64-bit virtual addressing
  - Original 360 was 24-bit; 370 was a 31-bit extension
- Superscalar, out-of-order
  - 12-wide issue
  - Up to 180 instructions in flight
- 16K-entry Branch Target Buffer
  - Very large buffer to support commercial workloads
- Four Levels of caches
  - 128KB L1 I-cache, 128KB L1 D-cache
  - 4MB L2 cache per core
  - 256MB shared on-chip L3 cache
  - 960MB shared off-chip L4 cache
- Up to 40TB of main memory per system



September 2019  
Image credit: IBM

# Instruction Set Architecture (ISA) versus Implementation

---

- ISA is the hardware/software interface
  - Defines set of programmer visible state
  - Defines data types
  - Defines instruction semantics (operations, sequencing)
  - Defines instruction format (bit encoding)
  - Examples: *MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM*
- Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z15 (c. 2019)
  - x86 implementations: *8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7, AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC*
  - MIPS implementations: *R2000, R4000, R10000, ...*
  - JVM: *HotSpot, PicoJava, ARM Jazelle, ...*

# Processor Performance

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

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

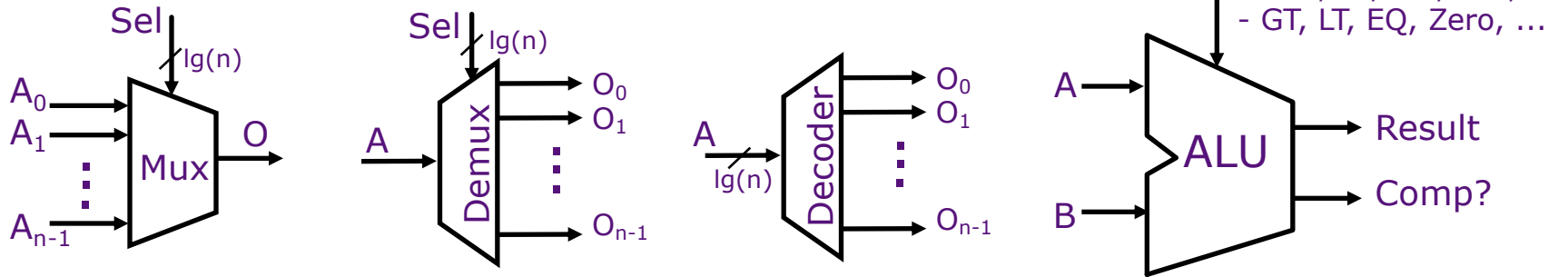
Microarchitecture	CPI	cycle time
Microcoded	> 1	short
Single-cycle unpipelined	1	long
Pipelined	1	short

rest of  
this lecture  
→

# Hardware Elements

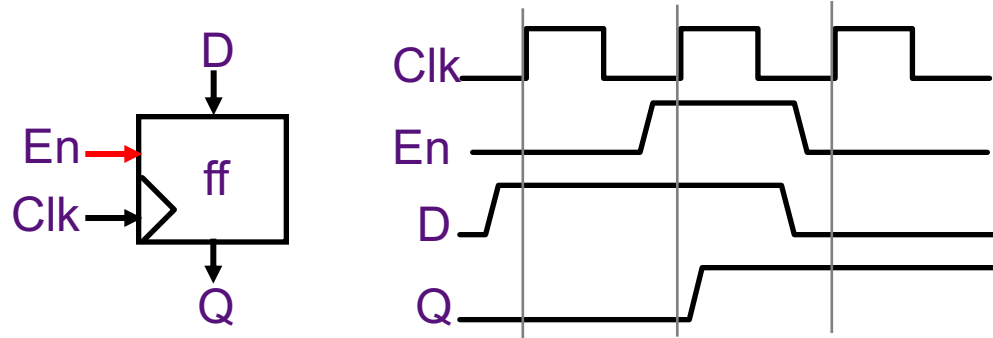
- Combinational circuits

- Mux, Demux, Decoder, ALU, ...



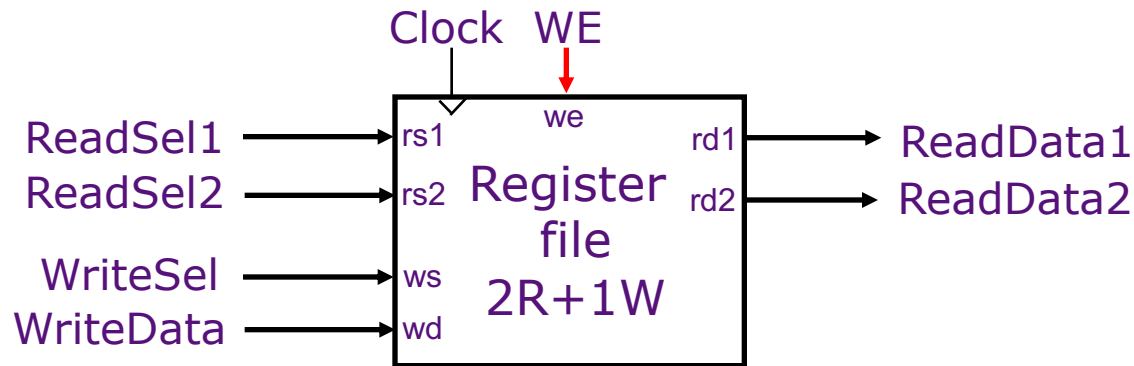
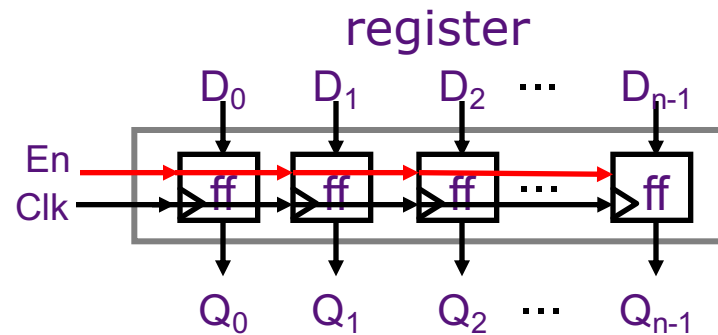
- Synchronous state elements

- Flipflop, Register, Register file, SRAM, DRAM



*Edge-triggered: Data is sampled at the rising edge*

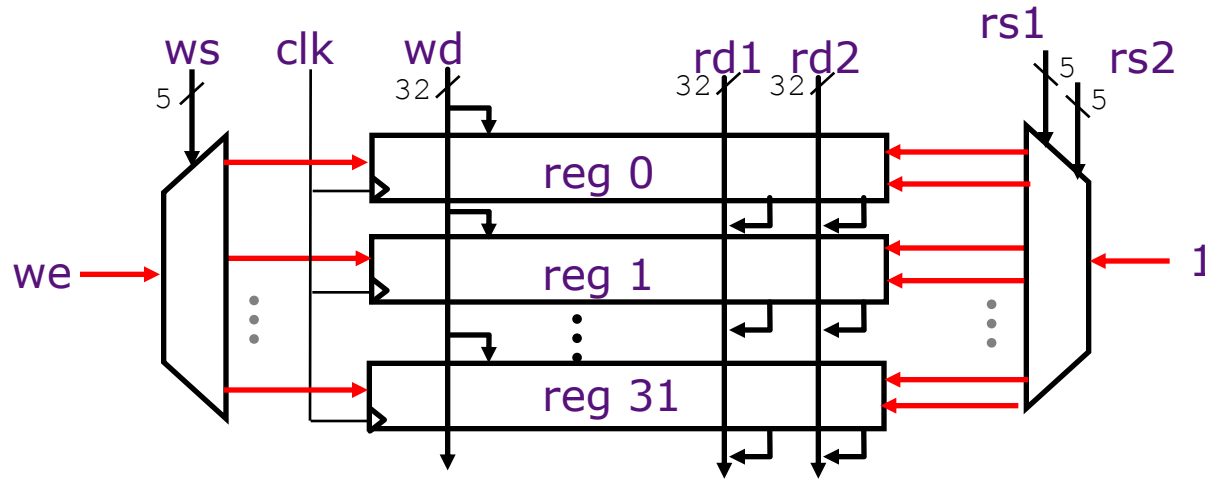
# Register Files



No timing issues when reading and writing the same register  
(writes happen at the end of the cycle)

# Register File Implementation

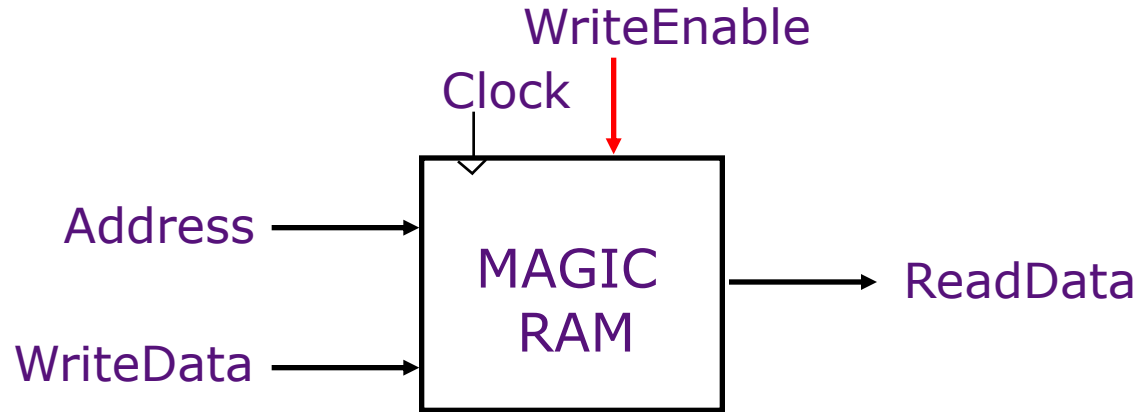
---



- Register files with a large number of ports are difficult to design
  - Area scales with ports<sup>2</sup>
  - Almost all Alpha instructions have exactly 2 register source operands
  - *Intel's Itanium GPR File has 128 registers with 8 read ports and 4 write 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

## 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

**All instructions are 32 bits**

# Instruction Execution

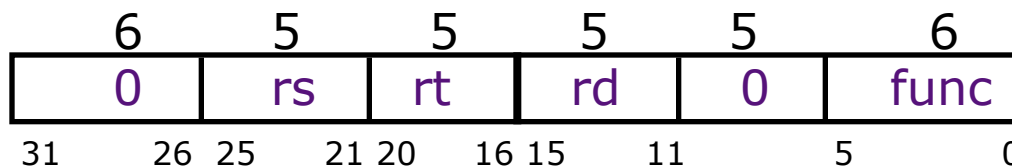
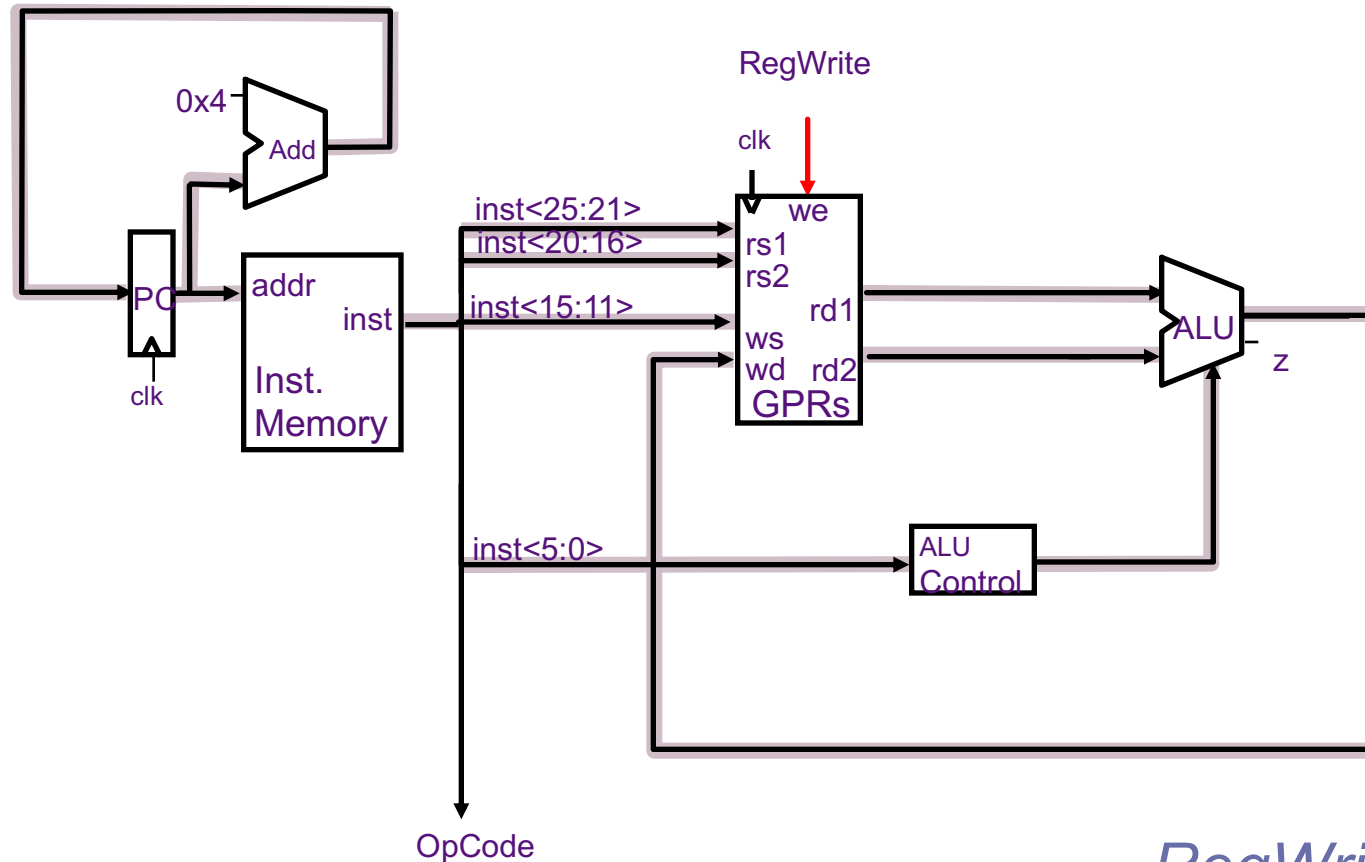
---

Execution of an instruction involves

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

And computing the address of the  
*next instruction (next PC)*

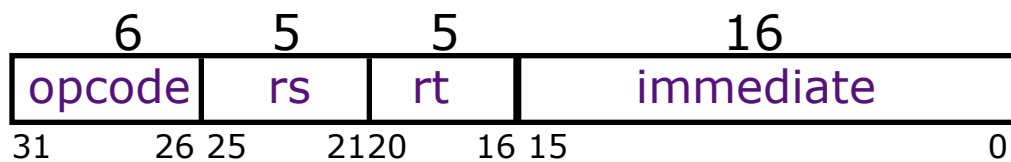
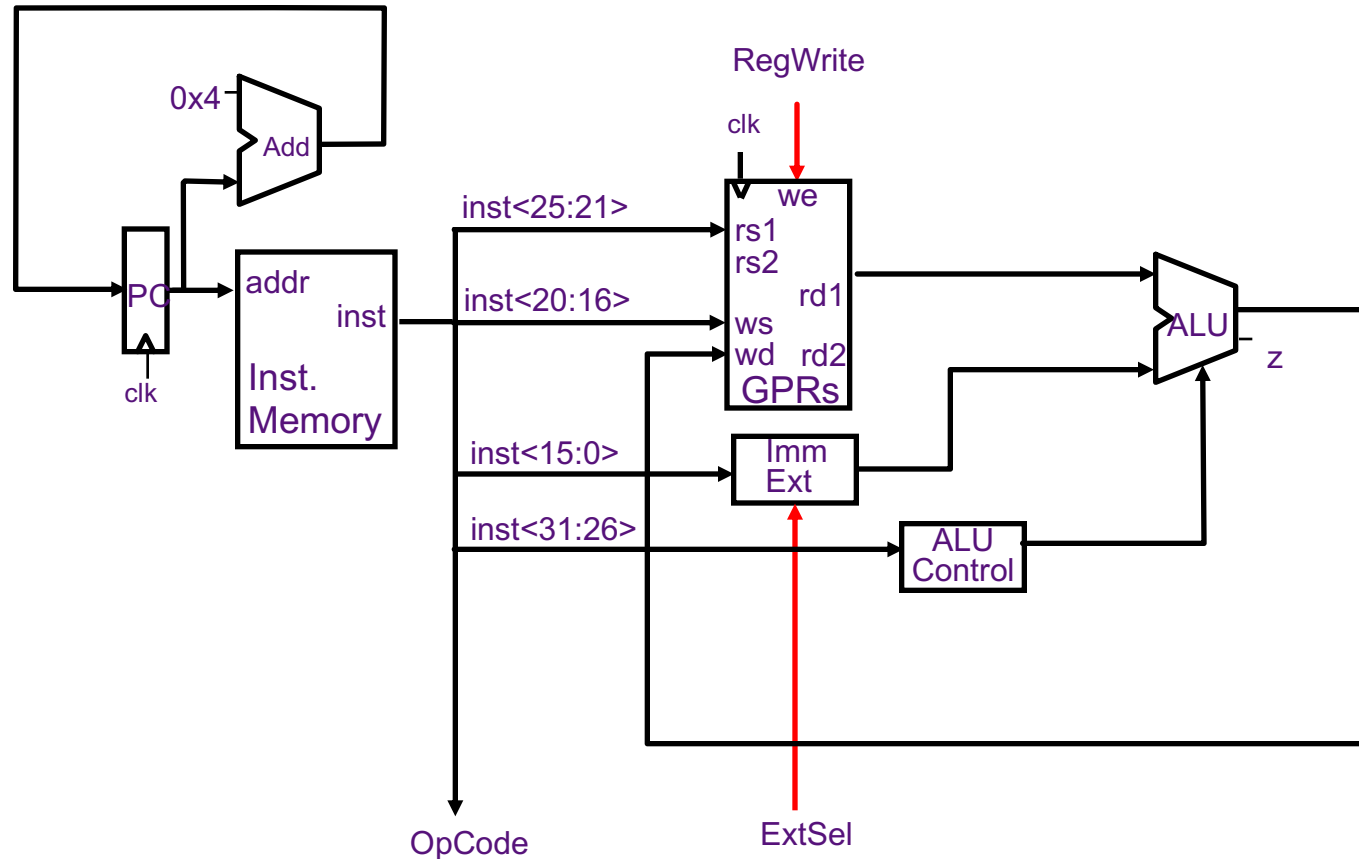
# Datapath: Reg-Reg ALU Instructions



*RegWrite Timing?*

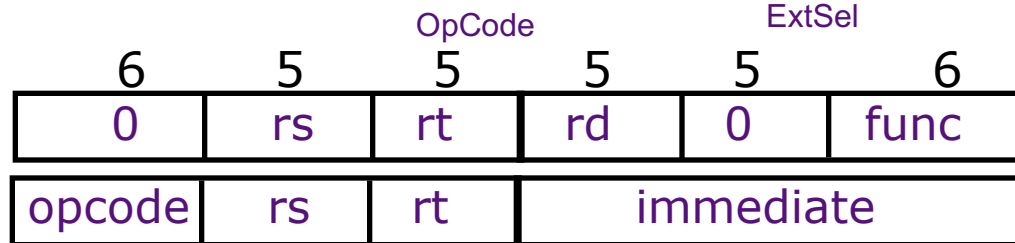
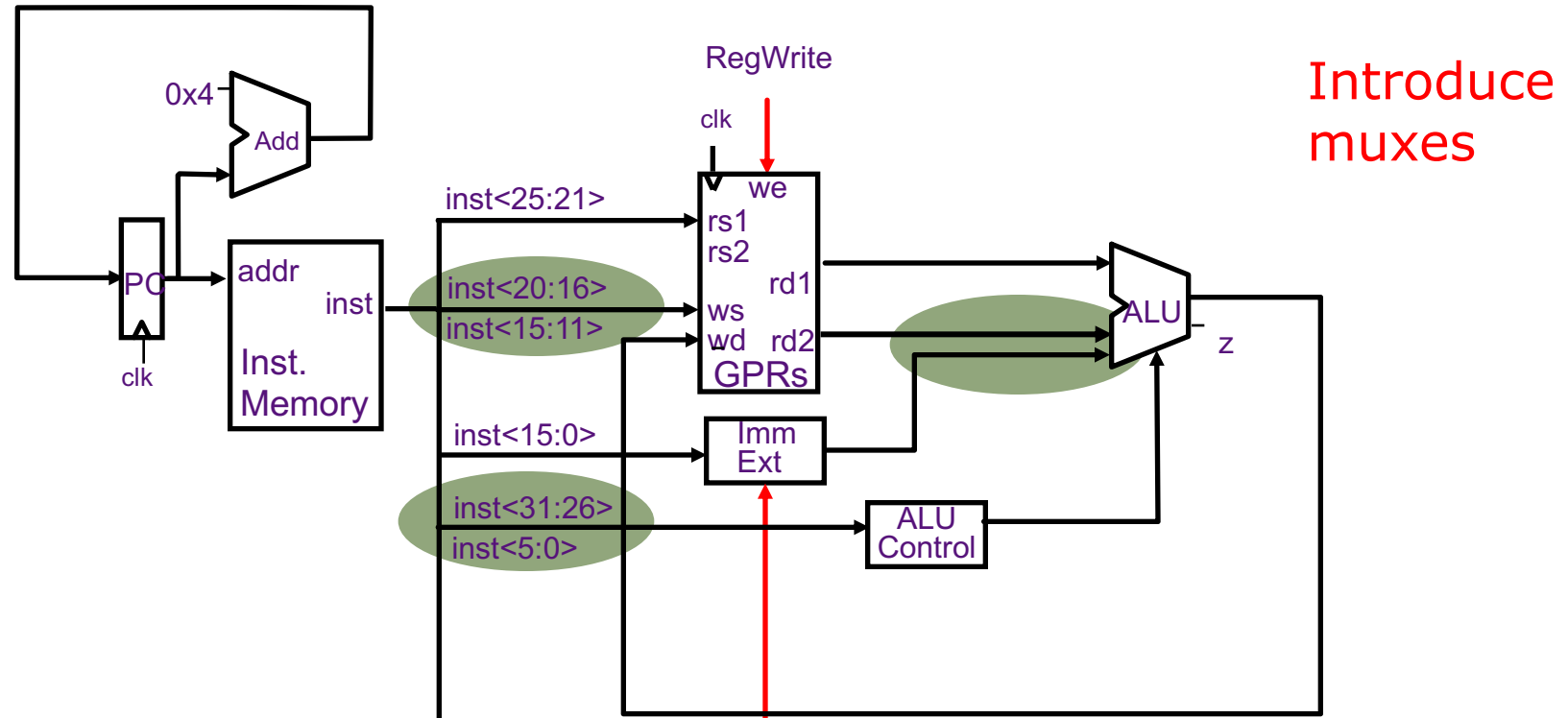
$$rd \leftarrow (rs) \text{ func } (rt)$$

# Datapath: Reg-Imm ALU Instructions



$rt \leftarrow (rs) \text{ op } \text{immediate}$

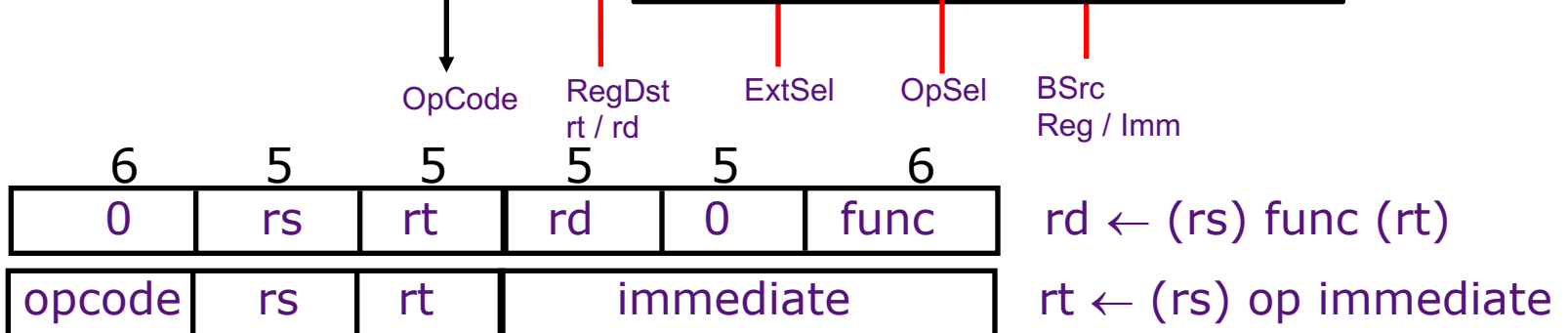
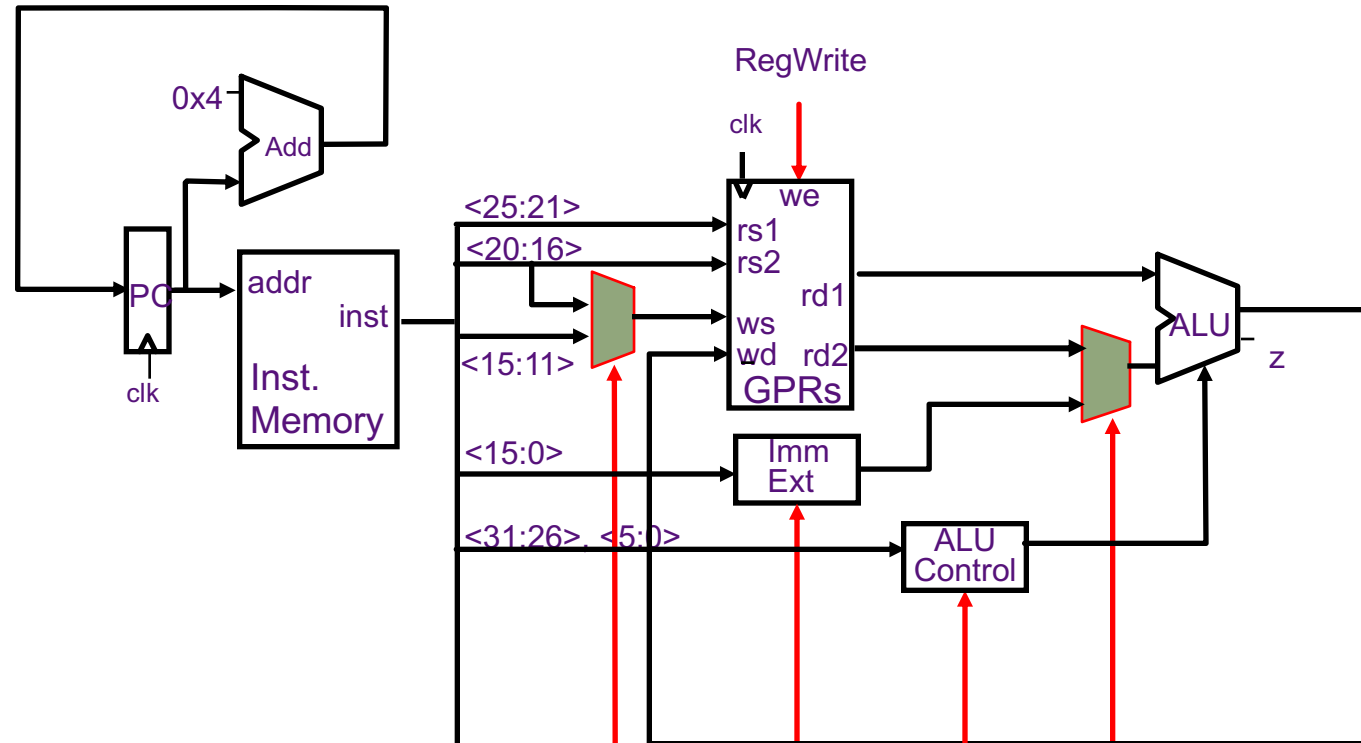
# Conflicts in Merging Datapath



$rd \leftarrow (rs) \text{ func } (rt)$

$rt \leftarrow (rs) \text{ op } \text{immediate}$

# Datapath for ALU Instructions



# Datapath for Memory Instructions

---

Should program and data memory be separate?

*Harvard style: separate* (Aiken and Mark 1 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

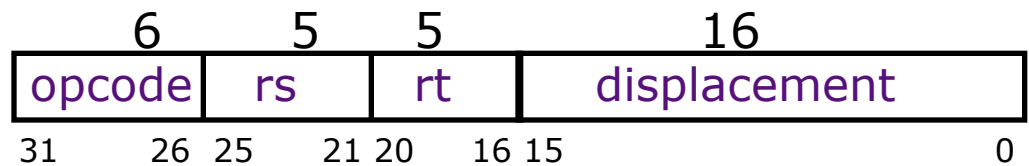
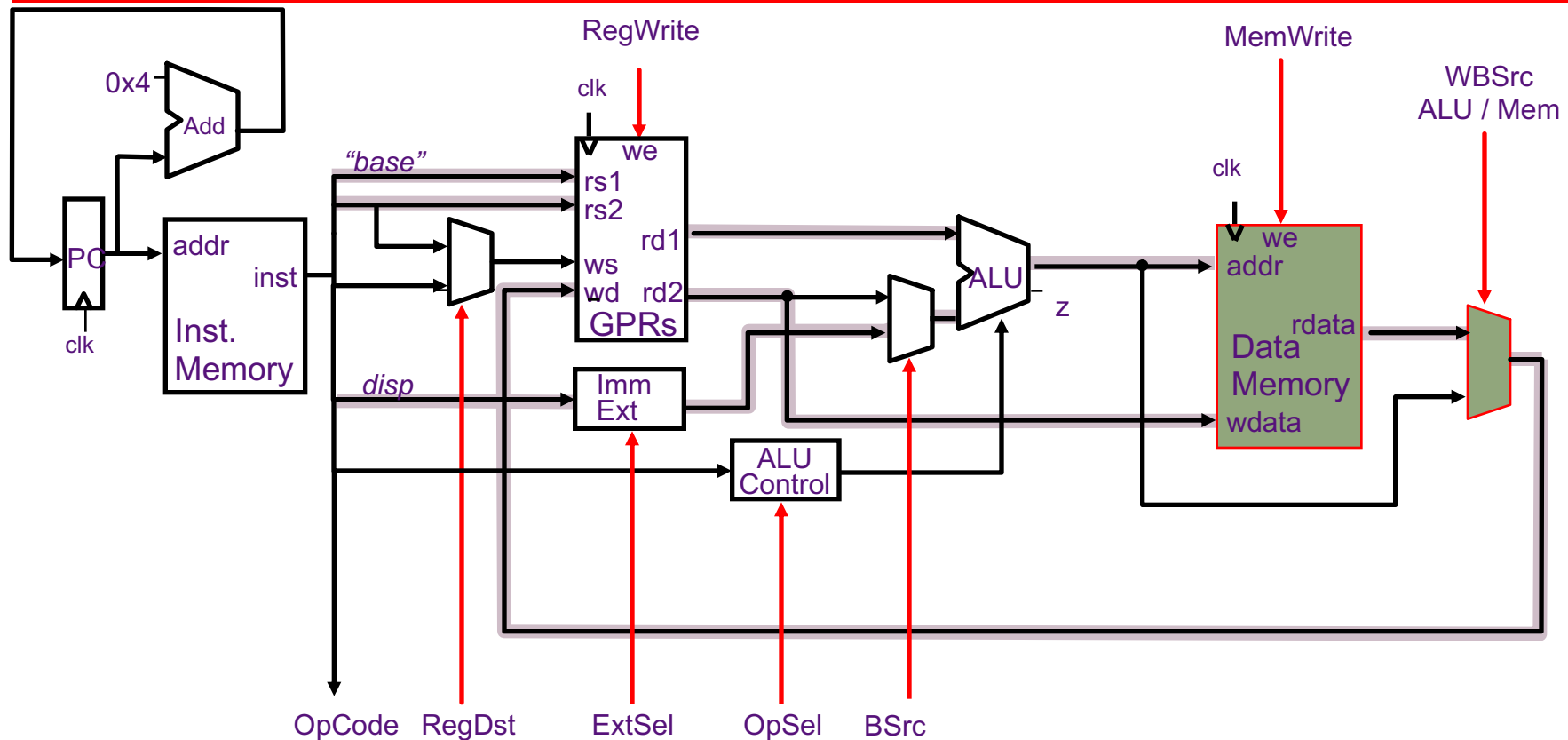
- Note:

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



# Load/Store Instructions

## Harvard Datapath



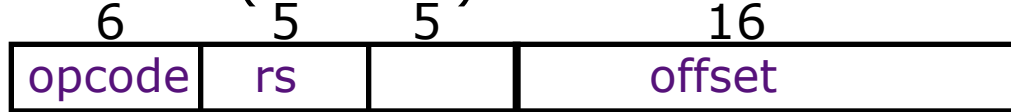
addressing mode  
(rs) + displacement

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



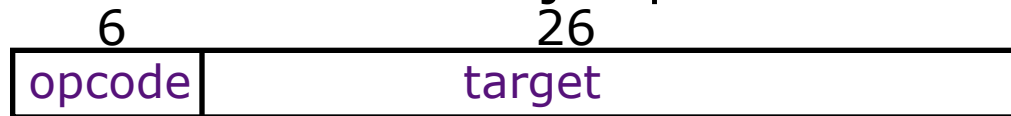
BEQZ, BNEZ

Unconditional register-indirect jumps



JR, JALR

Unconditional absolute jumps

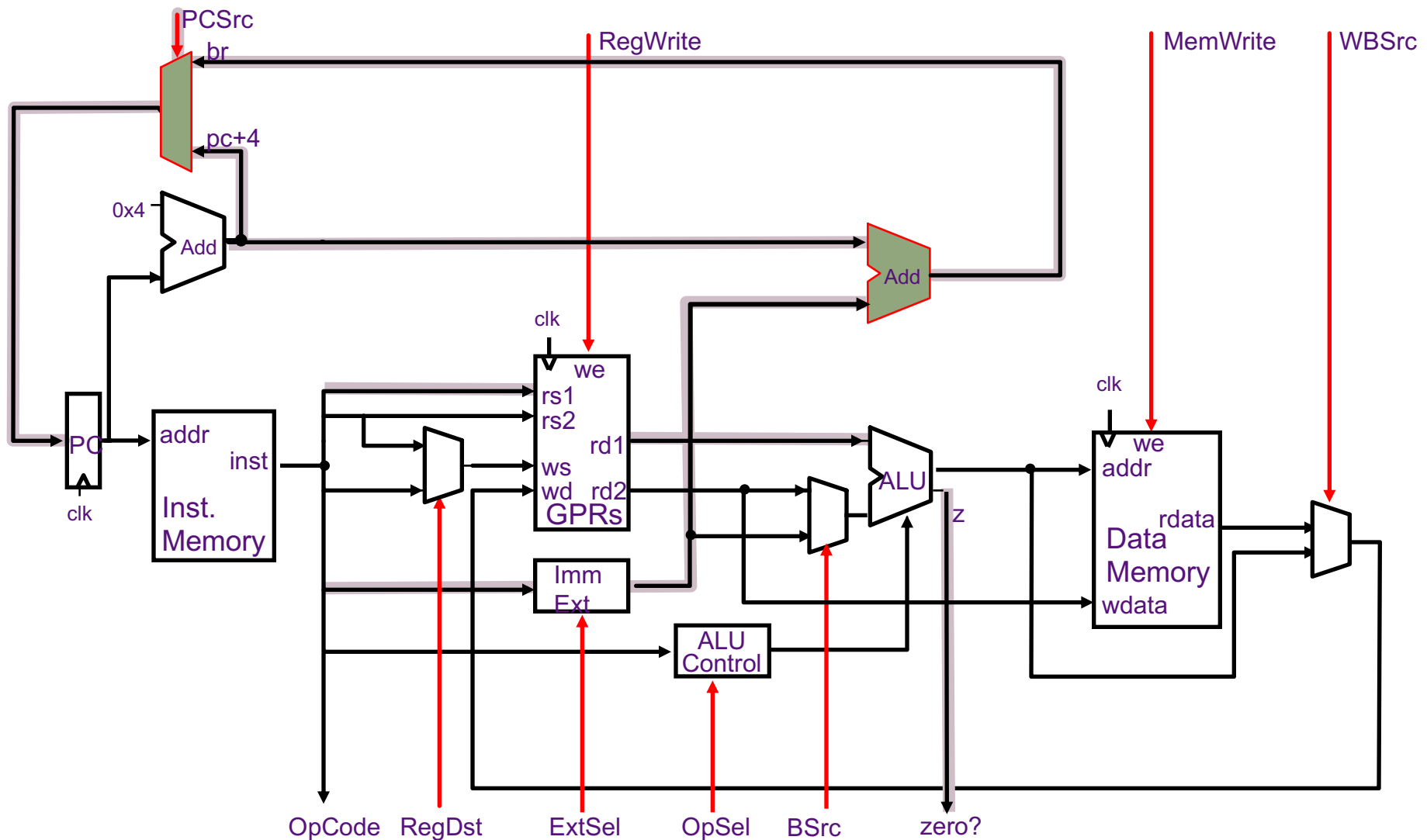


J, JAL

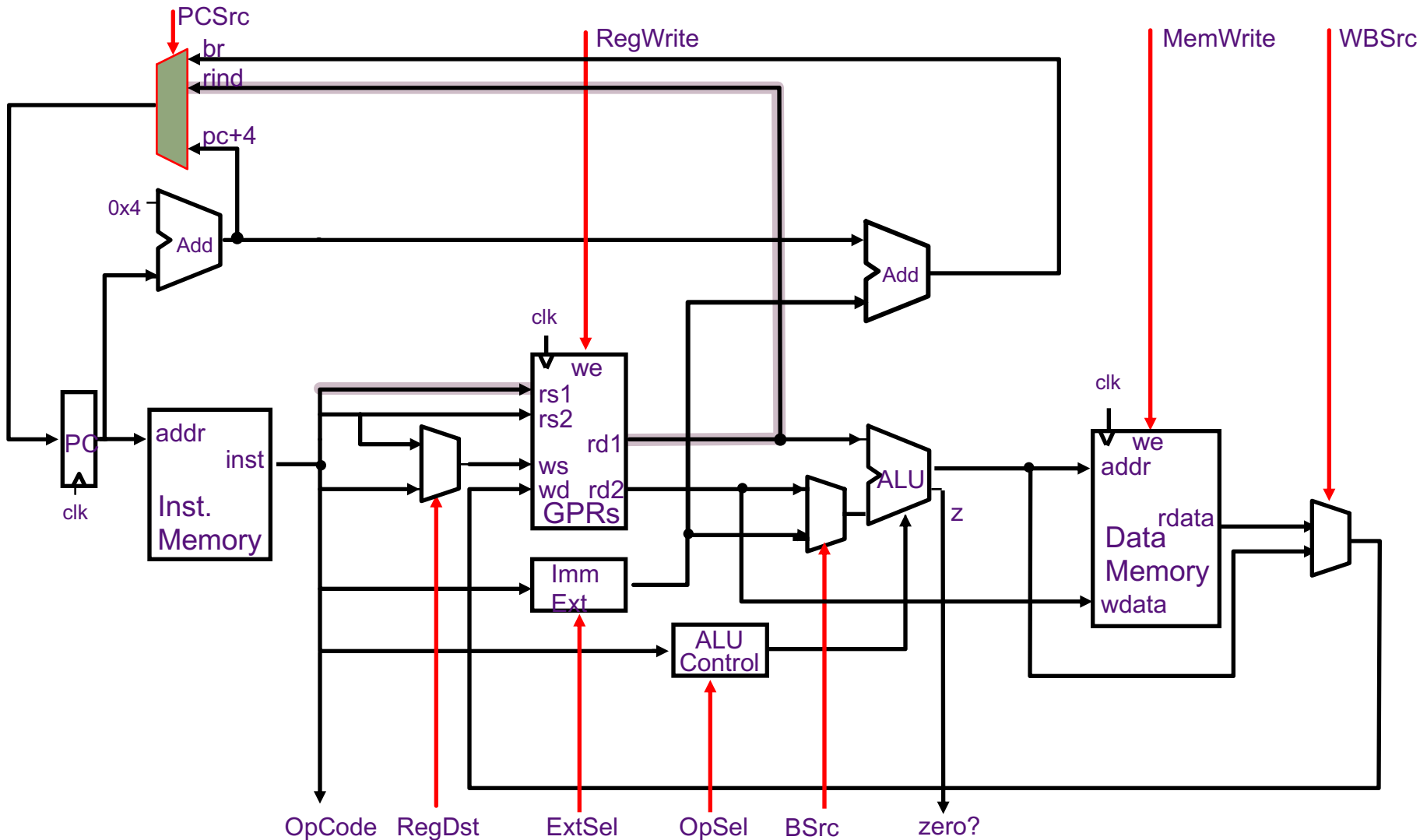
	Target PC	Condition
BEQZ	PC+4+offset*4	(rs)==0
BNEZ		(rs)!=0
JR, JALR	(rs)	Always taken
J, JAL	PC[31:28]  target*4	Always taken

- Jump-&-link stores PC+4 into the link register (R31)
- Control transfers are not delayed  
*we will worry about the branch delay slot later*

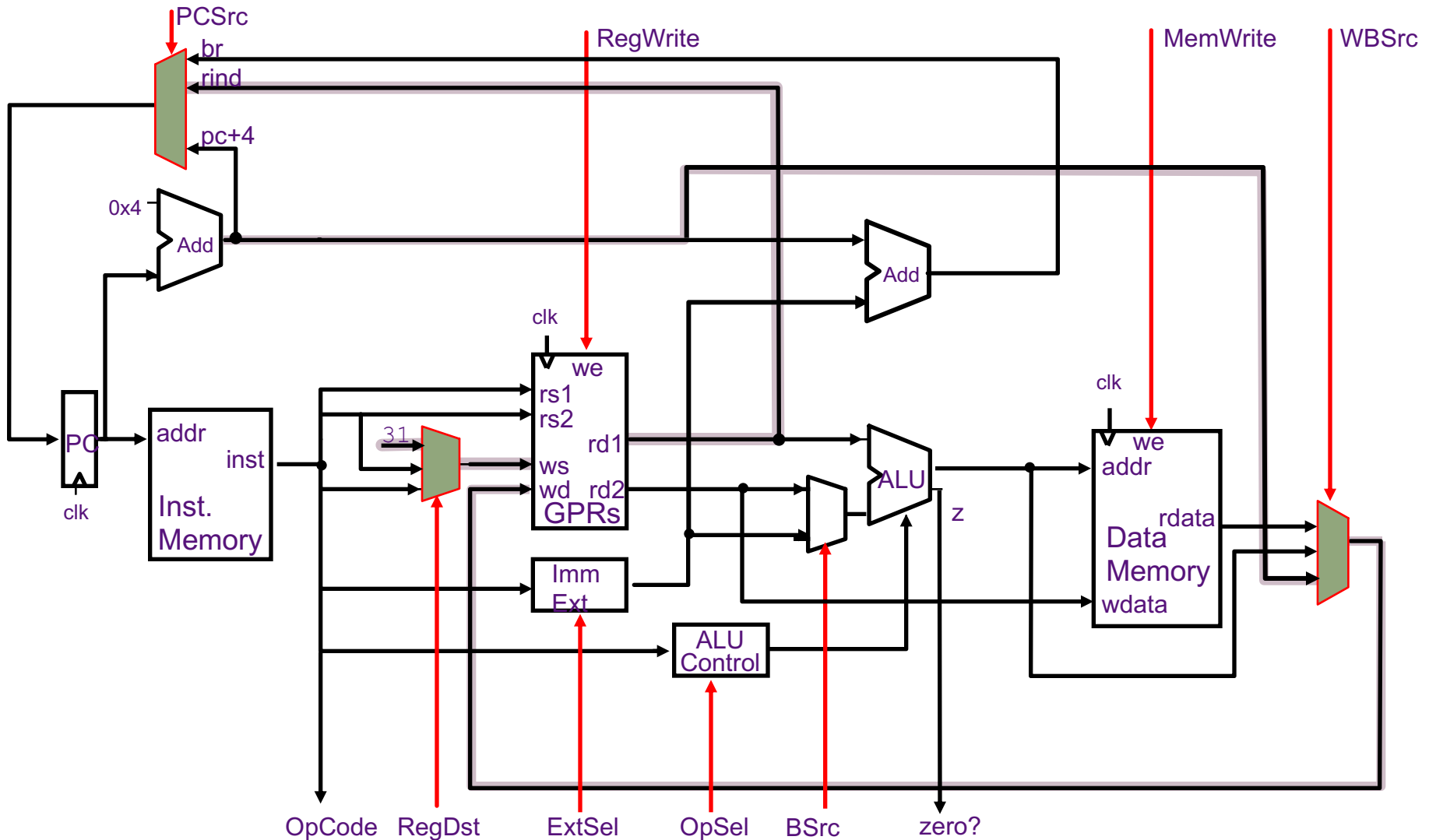
# Conditional Branches (BEQZ, BNEZ)



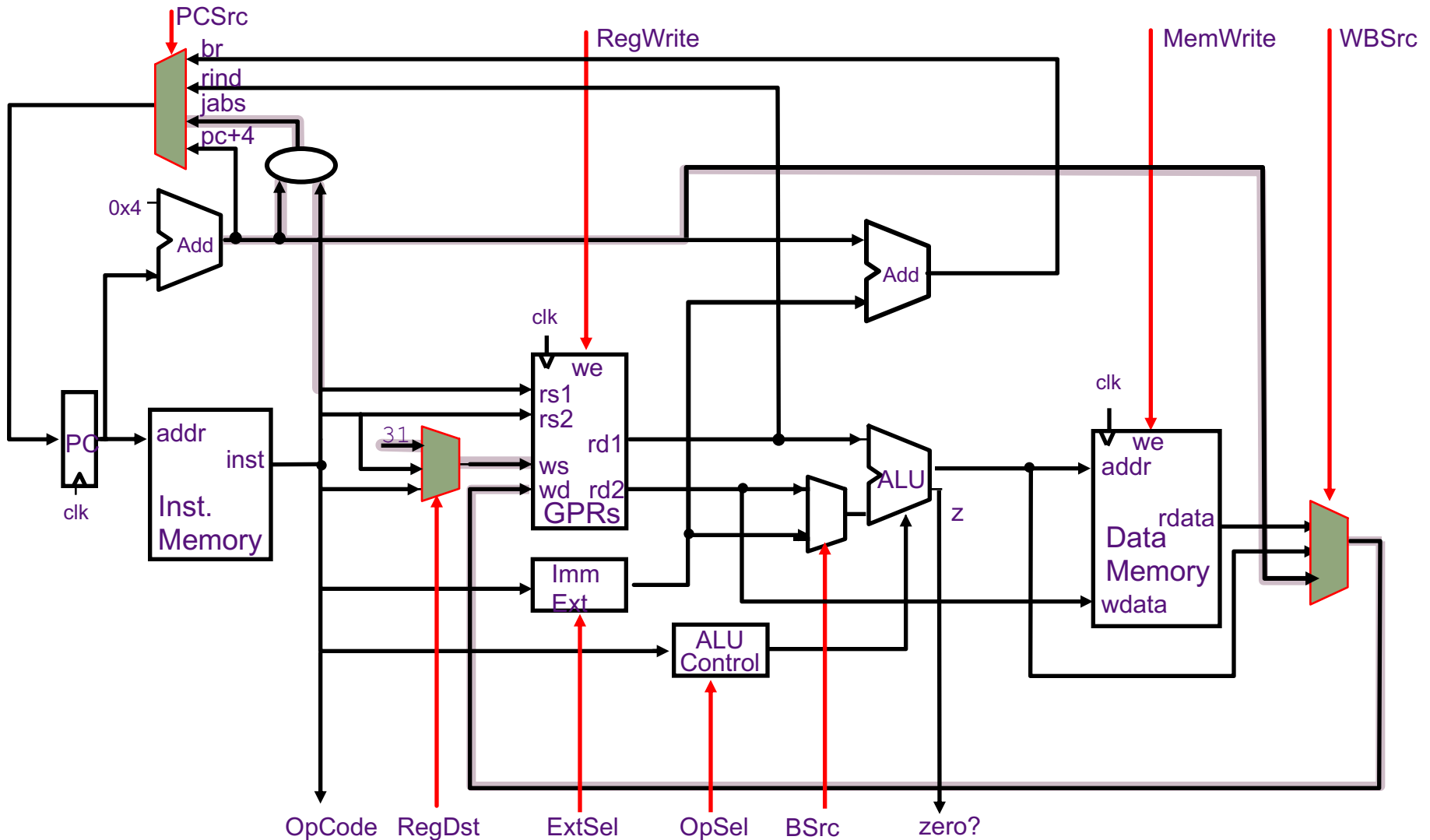
# Register-Indirect Jumps (JR)



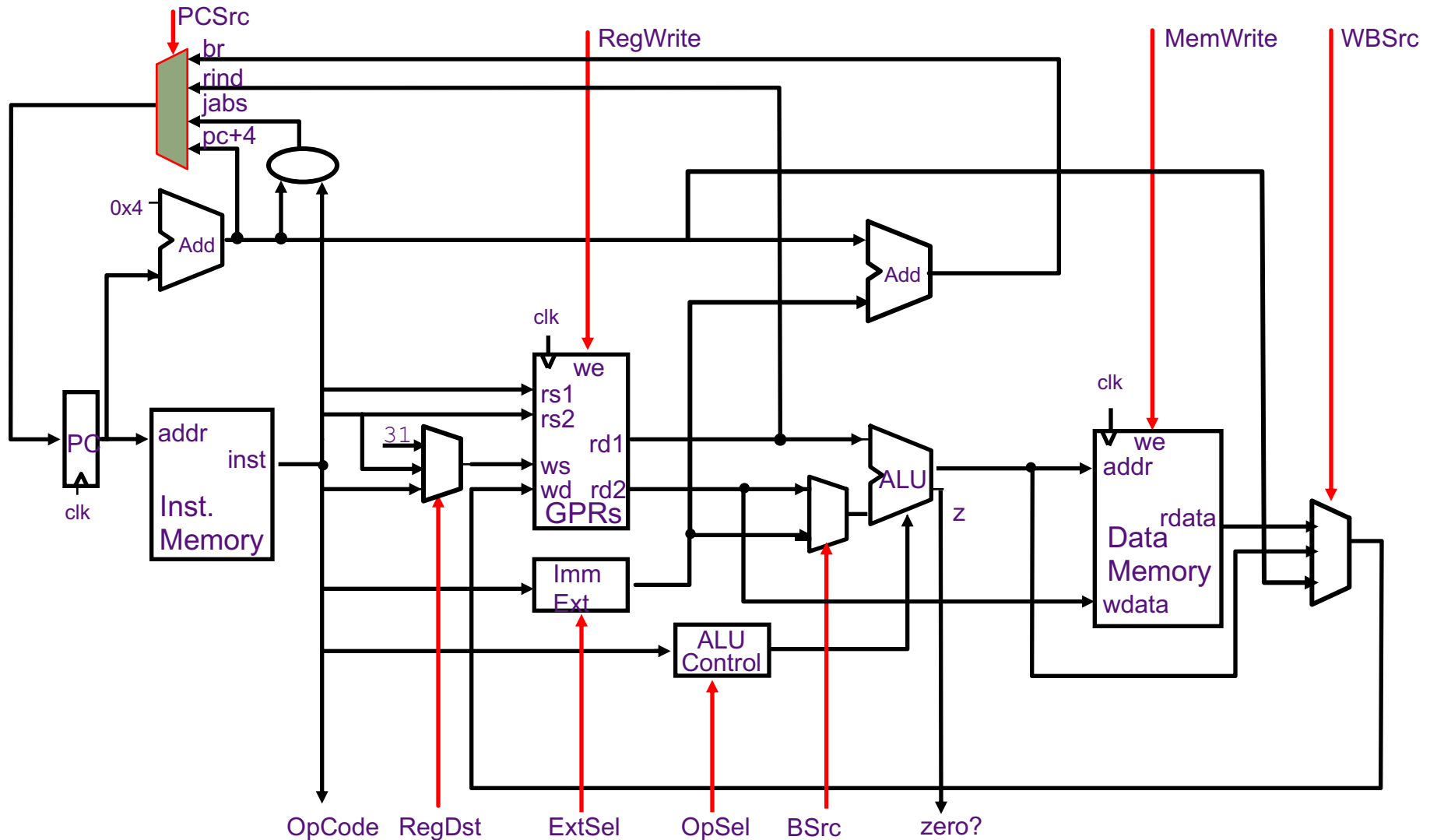
# Register-Indirect Jump-&-Link (JALR)



# Absolute Jumps (J, JAL)

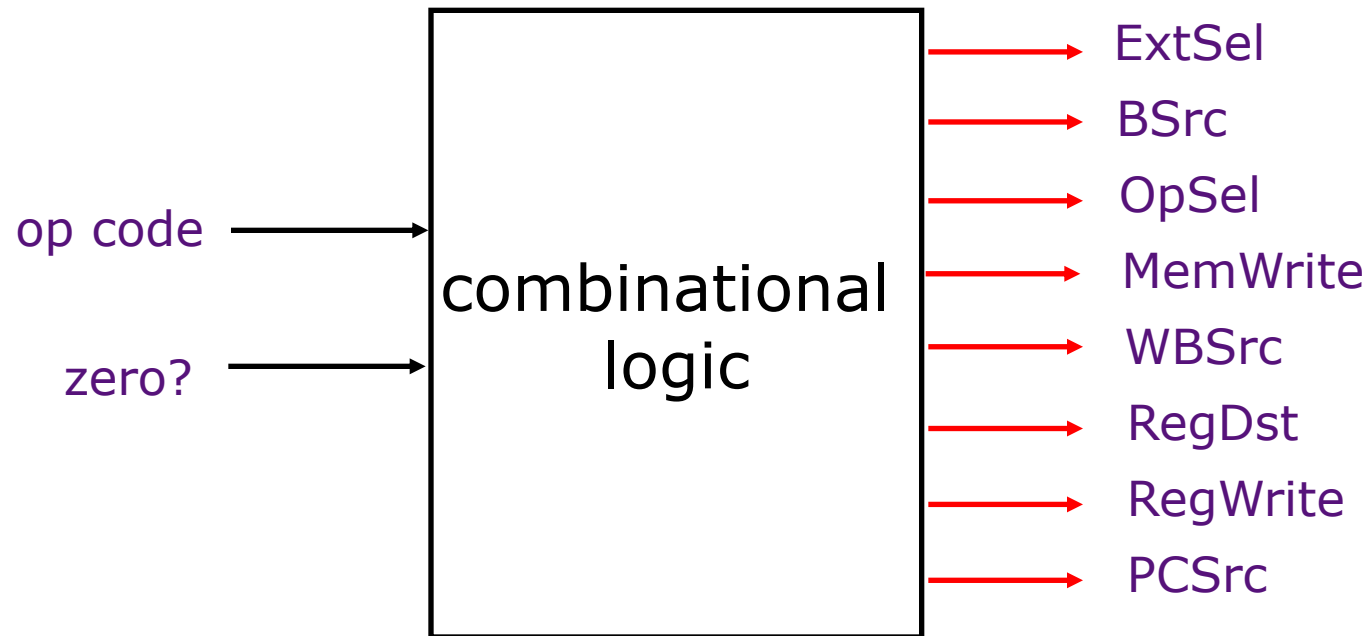


# Harvard-Style Datapath for MIPS



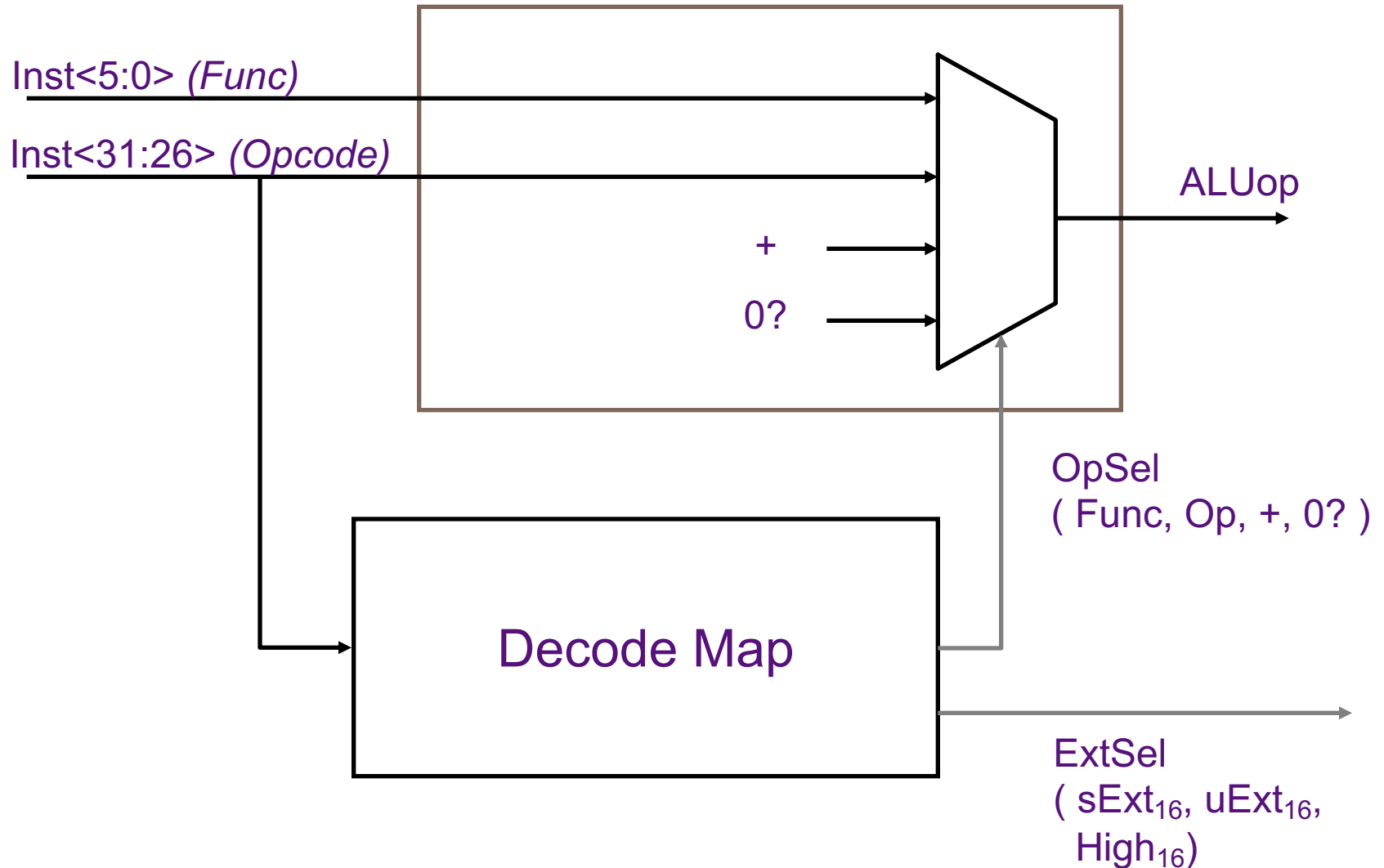
# Hardwired Control is pure Combinational Logic

---





# ALU Control & Immediate Extension



# Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no	yes	ALU	rd	pc+4
ALUi	sExt <sub>16</sub>	Imm	Op	no	yes	ALU	rt	pc+4
ALUiu	uExt <sub>16</sub>	Imm	Op	no	yes	ALU	rt	pc+4
LW	sExt <sub>16</sub>	Imm	+	no	yes	Mem	rt	pc+4
SW	sExt <sub>16</sub>	Imm	+	yes	no	*	*	pc+4
BEQZ <sub>z=0</sub>	sExt <sub>16</sub>	*	0?	no	no	*	*	br
BEQZ <sub>z=1</sub>	sExt <sub>16</sub>	*	0?	no	no	*	*	pc+4
J	*	*	*	no	no	*	*	jabs
JAL	*	*	*	no	yes	PC	R31	jabs
JR	*	*	*	no	no	*	*	rind
JALR	*	*	*	no	yes	PC	R31	rind

BSrc = Reg / Imm  
RegDst = rt / rd / R31

WBSrc = ALU / Mem / PC  
PCSrc = pc+4 / br / rind / jabs

# 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

# Princeton challenge

---

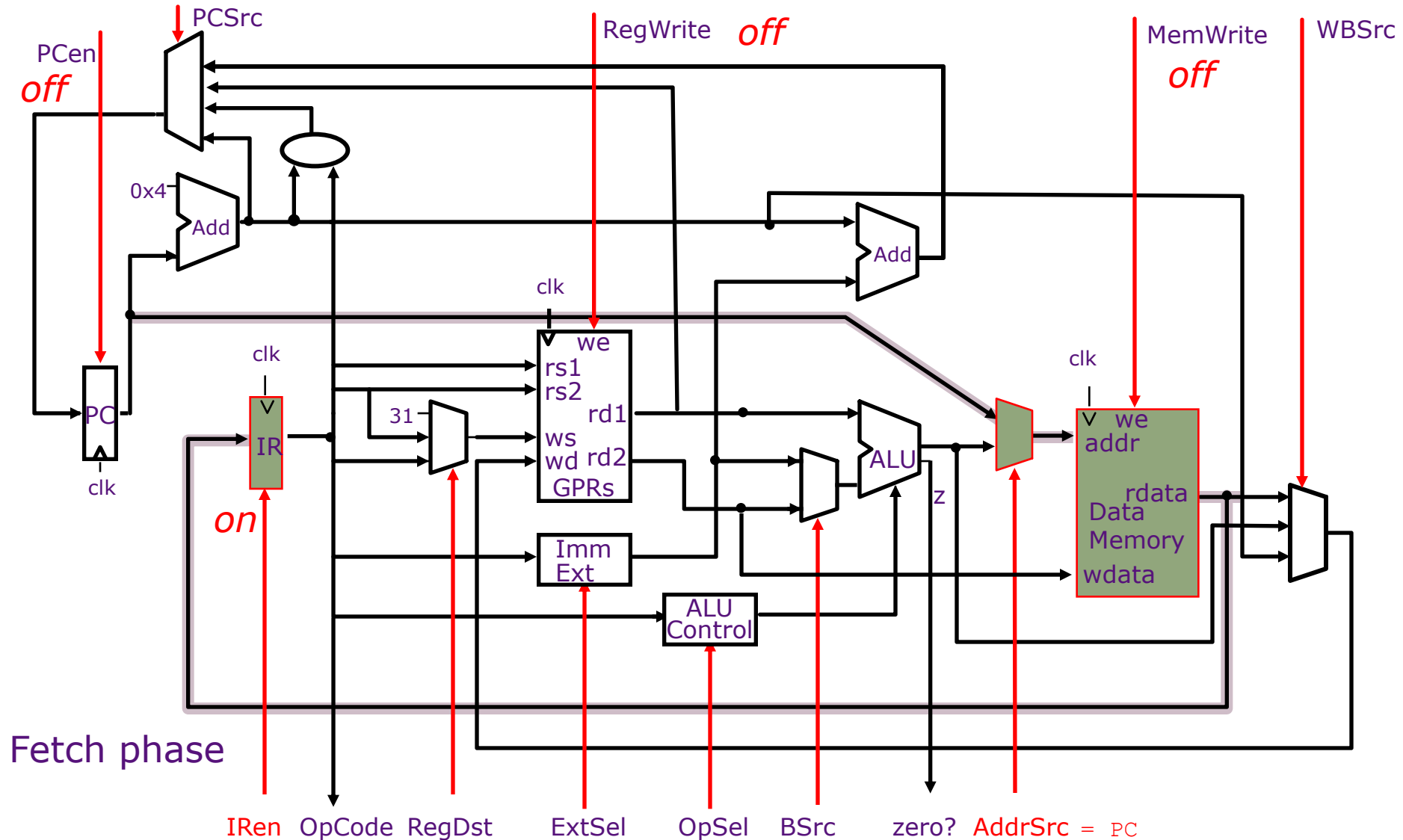
- What problem arises if instructions and data reside in the same memory?

At least the instruction fetch and a Load (or Store) cannot be executed in the same cycle

Structural hazard

# Princeton Microarchitecture

## Datapath & Control

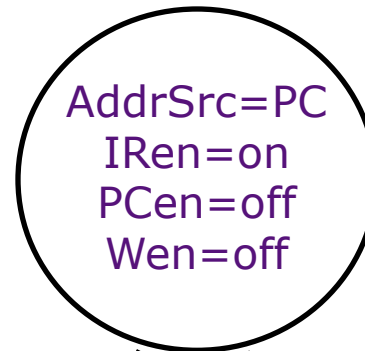


# Two-State Controller:

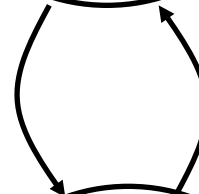
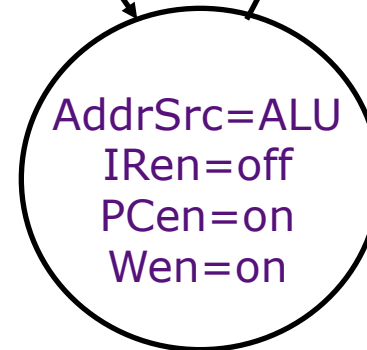
## *Princeton Architecture*

---

*fetch phase*



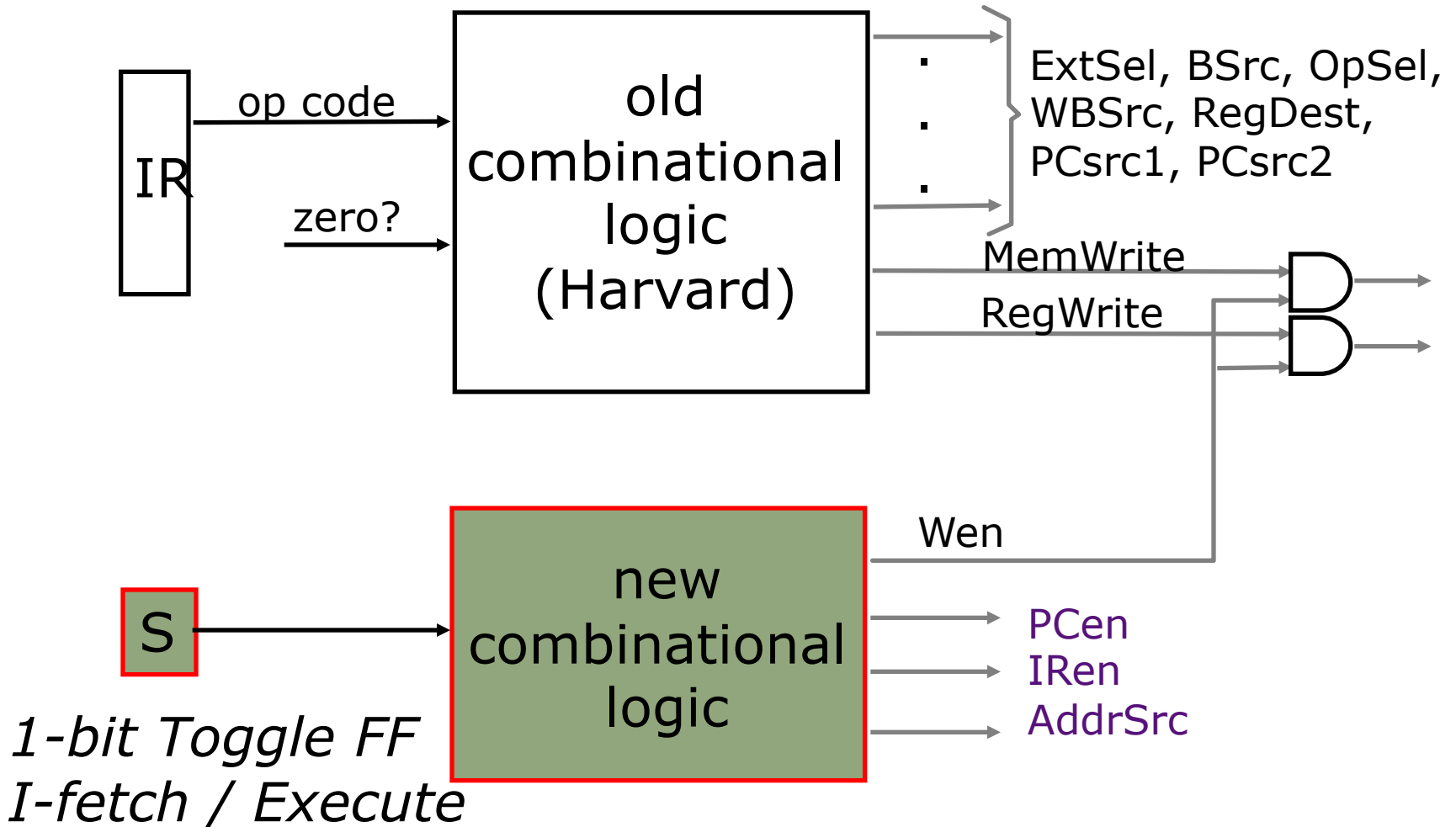
*execute phase*



A flipflop can be used to remember the phase

# Hardwired Controller:

## *Princeton Architecture*



# Clock Rate vs CPI

---

$$t_{\text{C-Princeton}} > \max \{t_M, t_{\text{RF}} + t_{\text{ALU}} + t_M + t_{\text{WB}}\}$$

$$t_{\text{C-Princeton}} > t_{\text{RF}} + t_{\text{ALU}} + t_M + t_{\text{WB}}$$

$$t_{\text{C-Harvard}} > t_M + t_{\text{RF}} + t_{\text{ALU}} + t_M + t_{\text{WB}}$$

Suppose  $t_M \gg t_{\text{RF}} + t_{\text{ALU}} + t_{\text{WB}}$

$$t_{\text{C-Princeton}} = 0.5 * t_{\text{C-Harvard}}$$

$$\text{CPI}_{\text{Princeton}} = 2$$

$$\text{CPI}_{\text{Harvard}} = 1$$

*No difference in performance!*

Is it possible to design a controller for the Princeton architecture with  $\text{CPI} < 2$  ?

*CPI = Clock cycles Per Instruction*

*Stay tuned!*