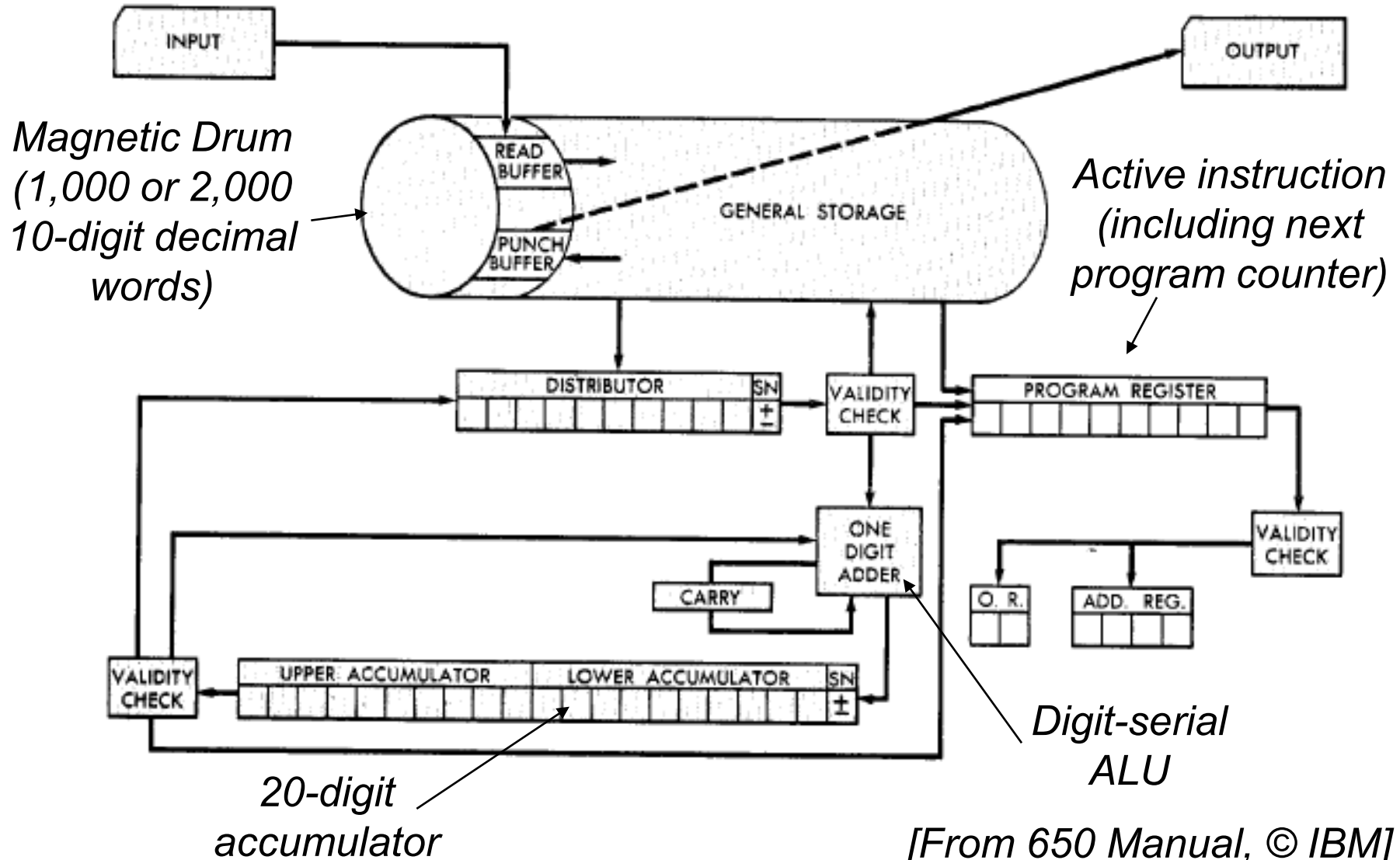# Instruction Set Architecture & Hardwired, Non-pipelined ISA Implementation

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

The IBM 650 (1953-4)

Magnetic Drum (1,000 or 2,000 10-digit decimal words)

Active instruction (including next program counter)

Digit-serial ALU

20-digit accumulator

[From 650 Manual, © IBM]

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)

	Model 30	. . .	Model 70
Memory Capacity	8K - 64 KB		256K - 512 KB
Memory Cycle	2.0µs	...	1.0µs
Datapath	8-bit		64-bit
Circuit Delay	30 nsec/level		5 nsec/level
Registers	in Main Store		in Transistor
Control Store	Read only 1µsec		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: Forty-Six years later... zEnterprise196 Microprocessor

- 1.4 billion transistors, Quad core design
- Up to 96 cores (80 visible to OS) in one multichip module
- 5.2 GHz, IBM 45nm SOI CMOS technology
- 64-bit virtual addressing
  - original 360 was 24-bit; 370 was a 31-bit extension
- Superscalar, out-of-order
  - Up to 72 instructions in flight
- Variable length instruction pipeline: 15-17 stages
- Each core has 2 integer units, 2 load-store units and 2 floating point units
- 8K-entry Branch Target Buffer
  - Very large buffer to support commercial workloads
- Four Levels of caches:
  - 64KB L1 I-cache, 128KB L1 D-cache
  - 1.5MB L2 cache per core
  - 24MB shared on-chip L3 cache
  - 192MB shared off-chip L4 cache
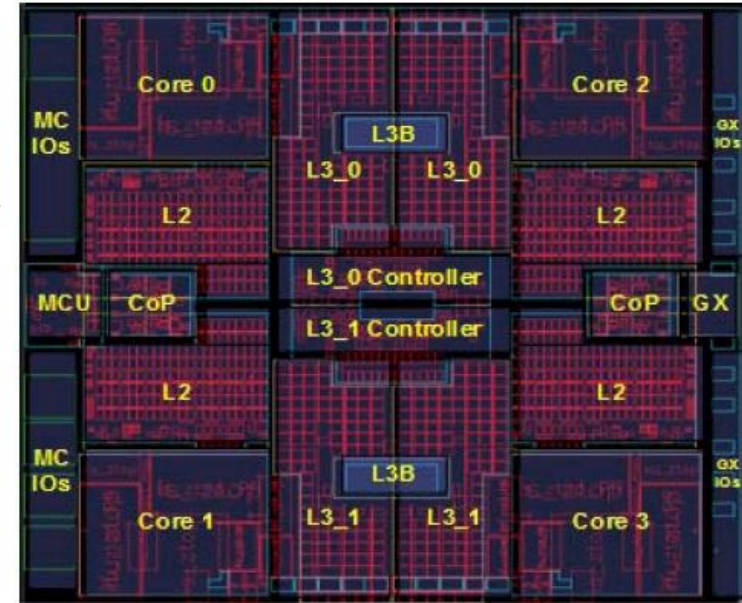
[September 2010]

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, Alpha, x86, IBM 360, VAX, ARM, JVM*

- Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), zEnterprise196 (c. 2010)
  - 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

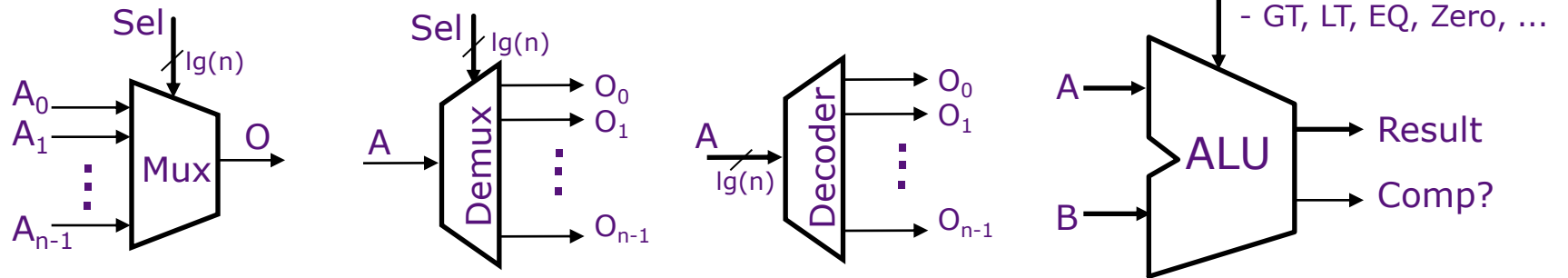– Time per cycle depends upon the microarchitecture and the base technology

rest of this lecture →

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

Hardware Elements

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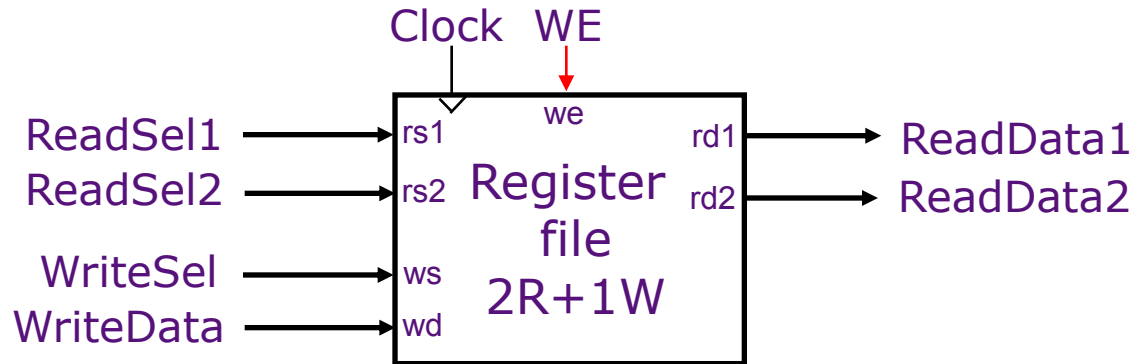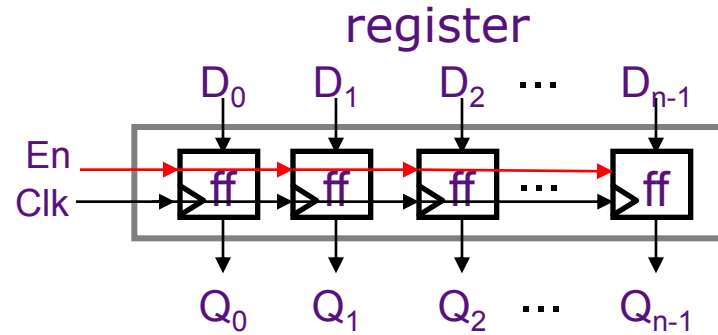 - Mux, Demux, Decoder, ALU, ...

OpSelect
- Add, Sub, ...
- And, Or, Xor, Not, ...
- GT, LT, EQ, Zero, ...

Sel $/lg(n)$

A_0
A_1
$\vdots$
A_{n-1}
Mux
O

Sel $/lg(n)$

A
Demux
O_0
O_1
$\vdots$
O_{n-1}

A $/lg(n)$
Decoder
O_0
O_1
$\vdots$
O_{n-1}

A
B
ALU
Result
Comp?

- ## Synchronous state elements
  - Flipflop, Register, Register file, SRAM, DRAM

D
En
Clk
ff
Q

Clk
En
D
Q
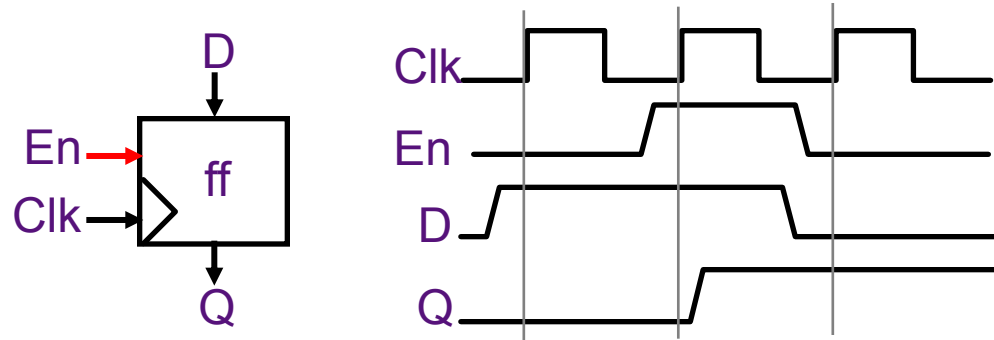
Edge-triggered: Data is sampled at the rising edge

Register Files

register

D_0 D_1 D_2 $\cdots$ D_{n-1}

En

Clk

ff ff ff $\cdots$ ff

Q_0 Q_1 Q_2 $\cdots$ Q_{n-1}

Clock WE

we

ReadSel1 → rs1

ReadSel2 → rs2

WriteSel → ws

WriteData → wd

Register file 2R+1W

rd1 → ReadData1

rd2 → ReadData2

No timing issues in reading a selected register

Register File Implementation
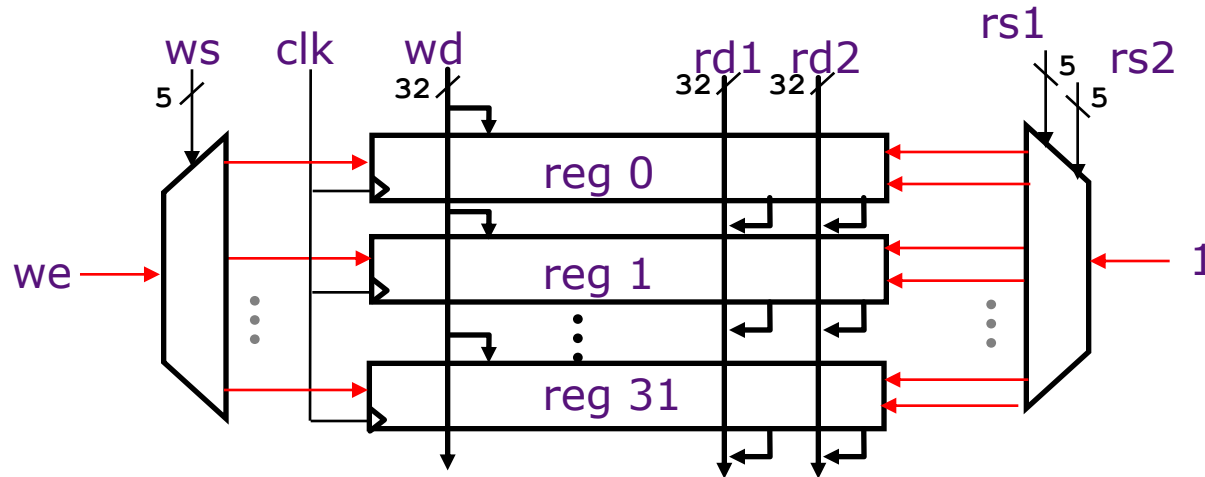
- ## Register files with a large number of ports are difficult to design

  - Area scales with ports^2
  - 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

WriteEnable

Clock

Address → | MAGIC RAM | → ReadData
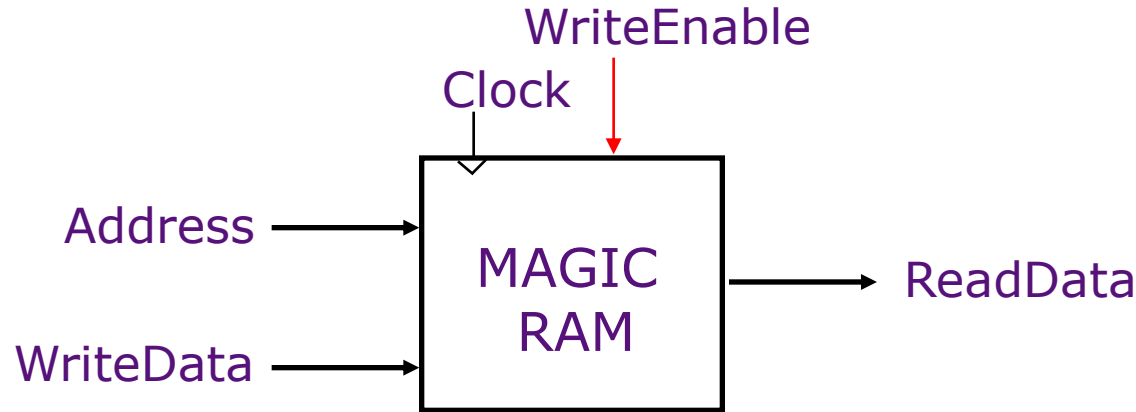
WriteData →

- ## 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
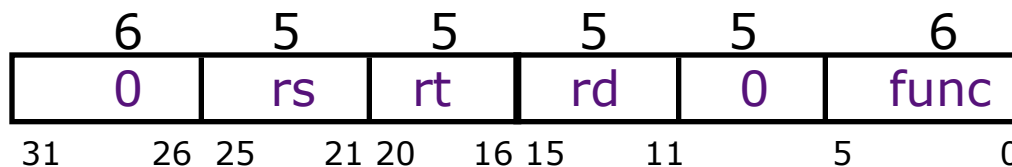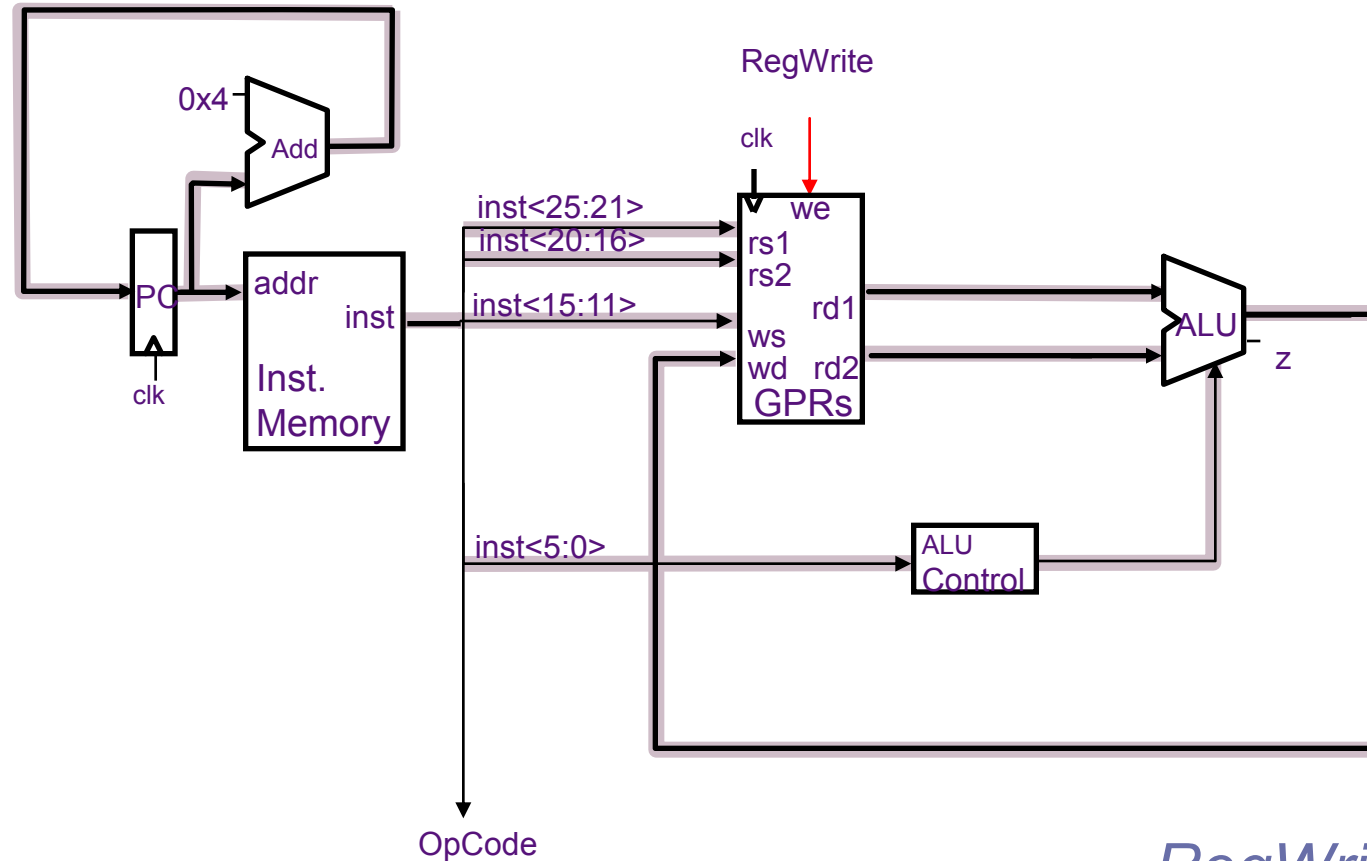
All instructions are 32 bits

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
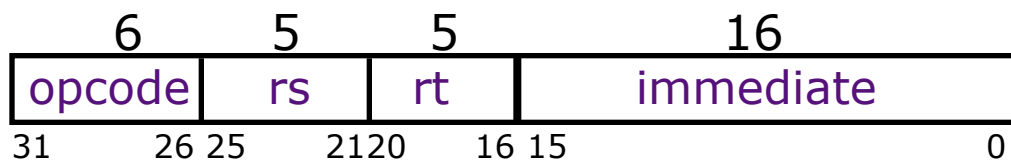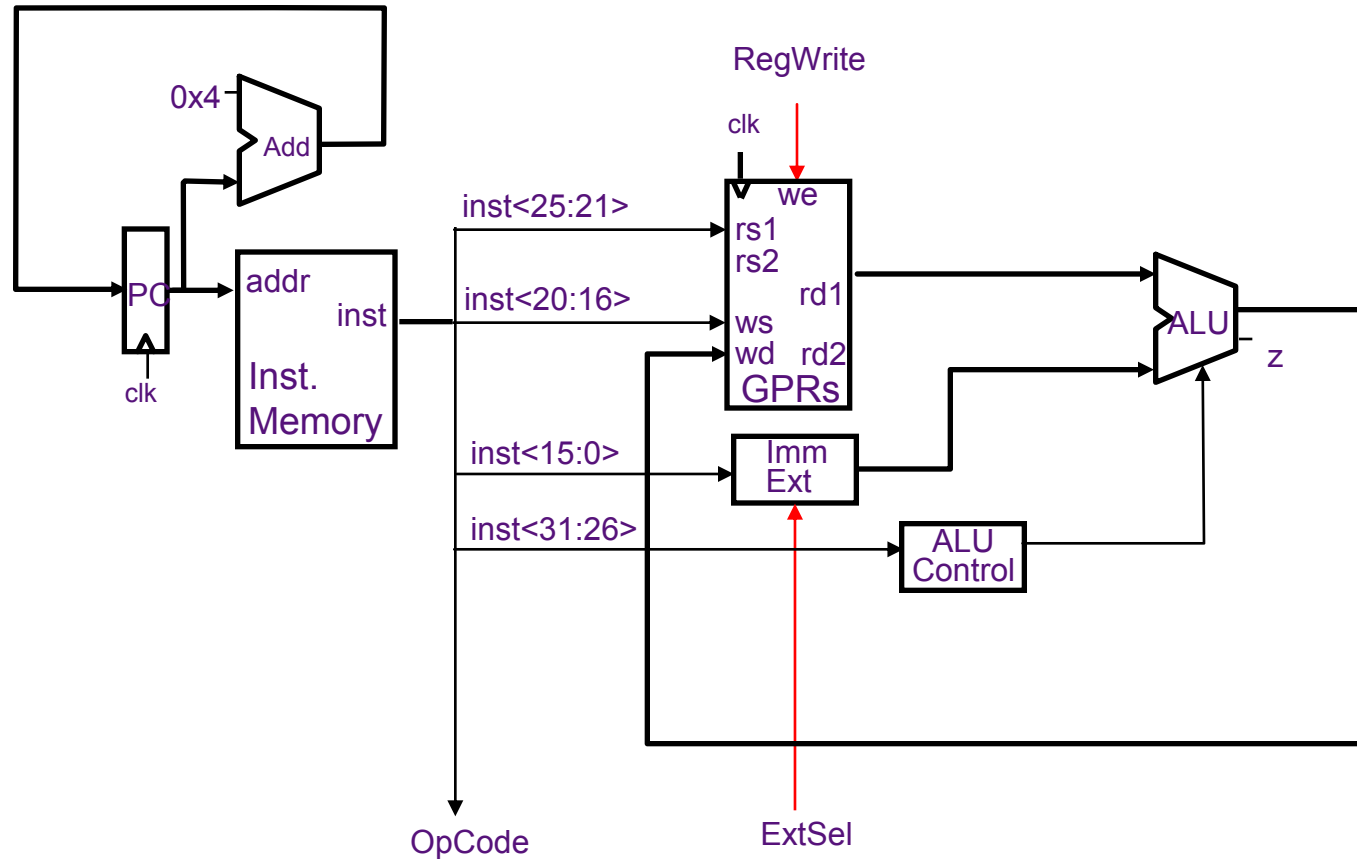*next instruction (next PC)*

Datapath: Reg-Reg ALU Instructions
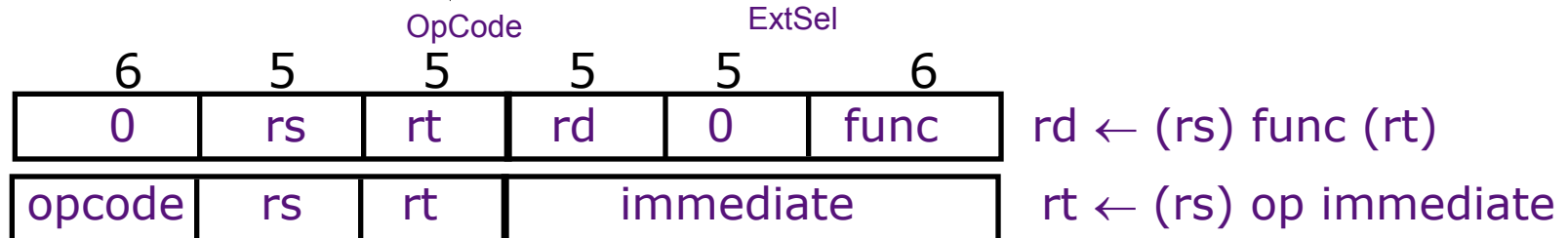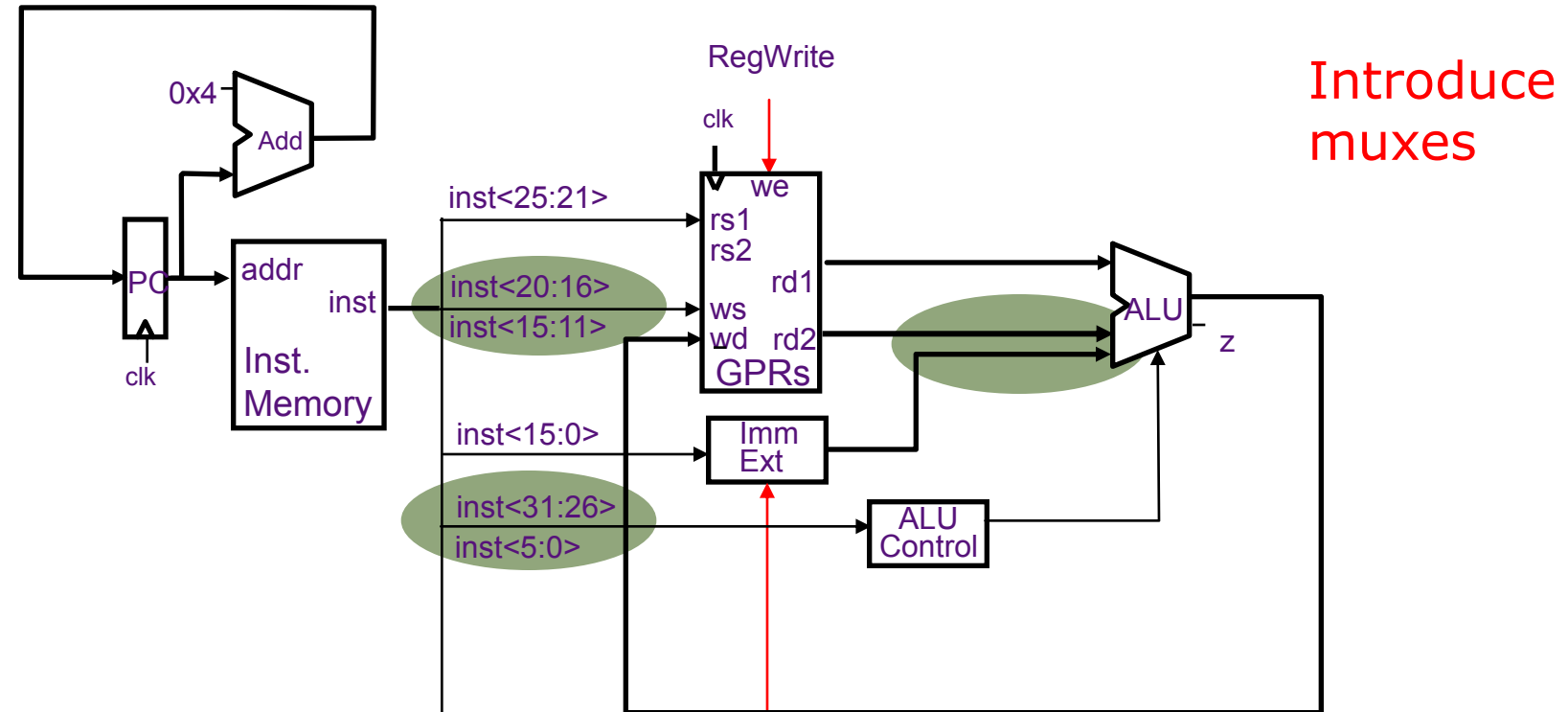

RegWrite Timing?

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

6	5	5	16
opcode	rs	rt	immediate

31   26 25   2120   16 15   0

$rt \leftarrow (rs)$ 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 |
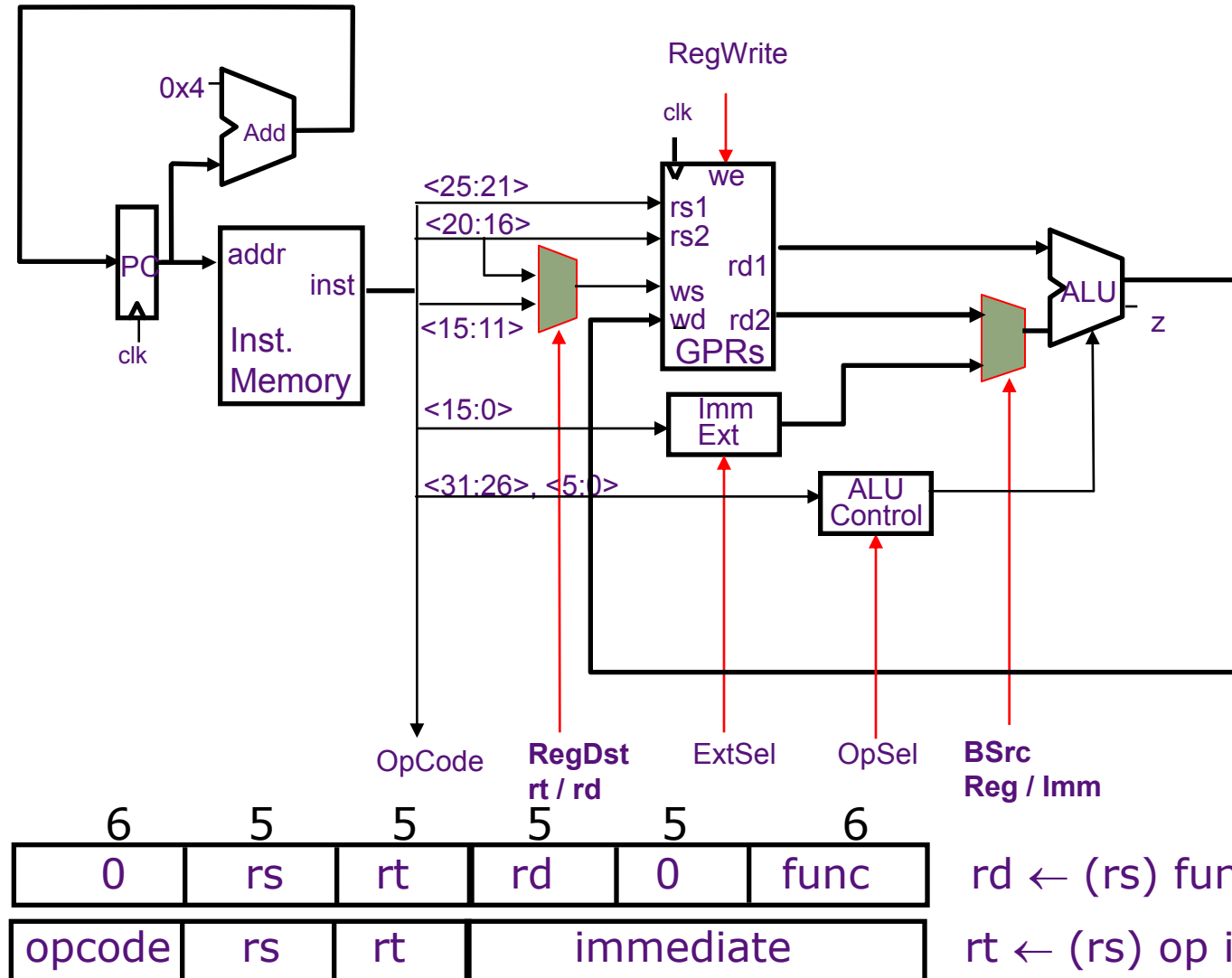
Datapath for Memory Instructions

Should program and data memory be separate?

Harvard style: separate (Aiken and Mark 1 influence)
- read-only program memory
- read/write data memory

- Note:
  There must be a way to load the program memory

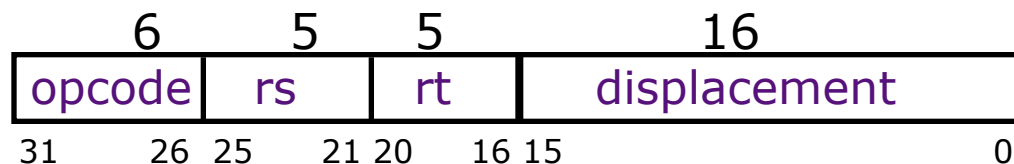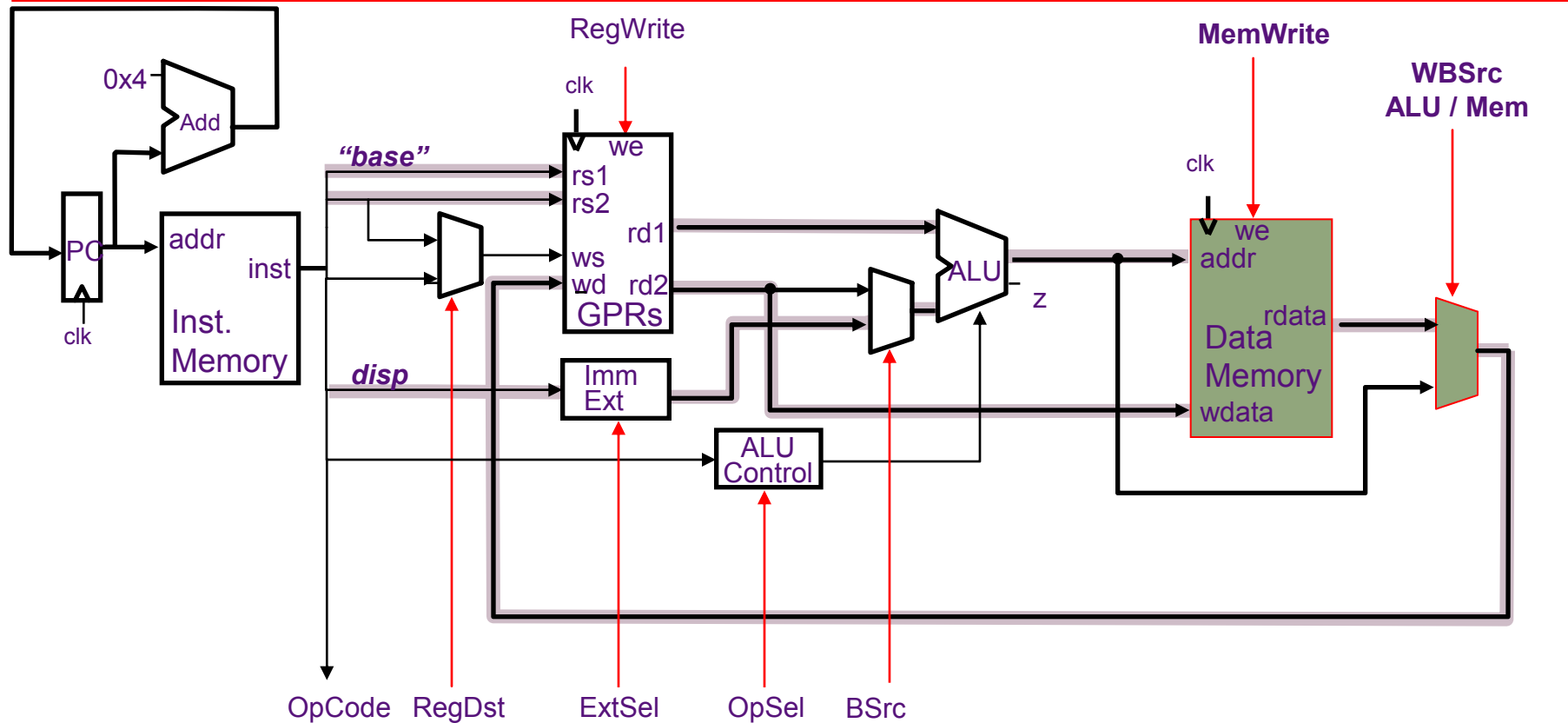*Princeton style: the same* (von Neumann's influence)
- single read/write memory for program and data

- Note:
  Executing a Load or Store instruction requires accessing the memory more than once
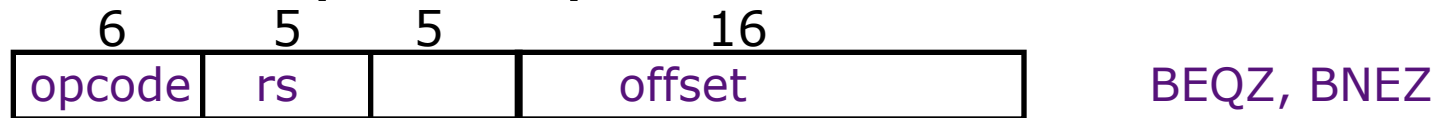
Load/Store Instructions
Harvard Datapath

6	5	5	16	addressing mode
opcode	rs	rt	displacement	(rs) + displacement

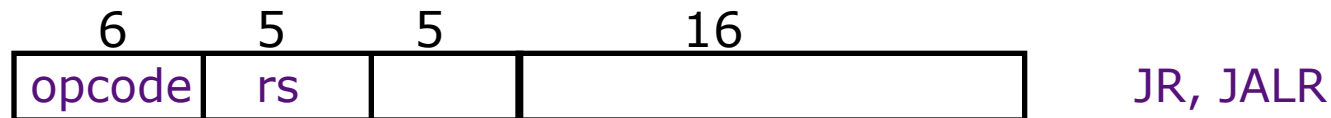31   26 25   21 20   16 15   0

rs is the base register

rt is the destination of a Load or the source for a Store

MIPS Control Instructions
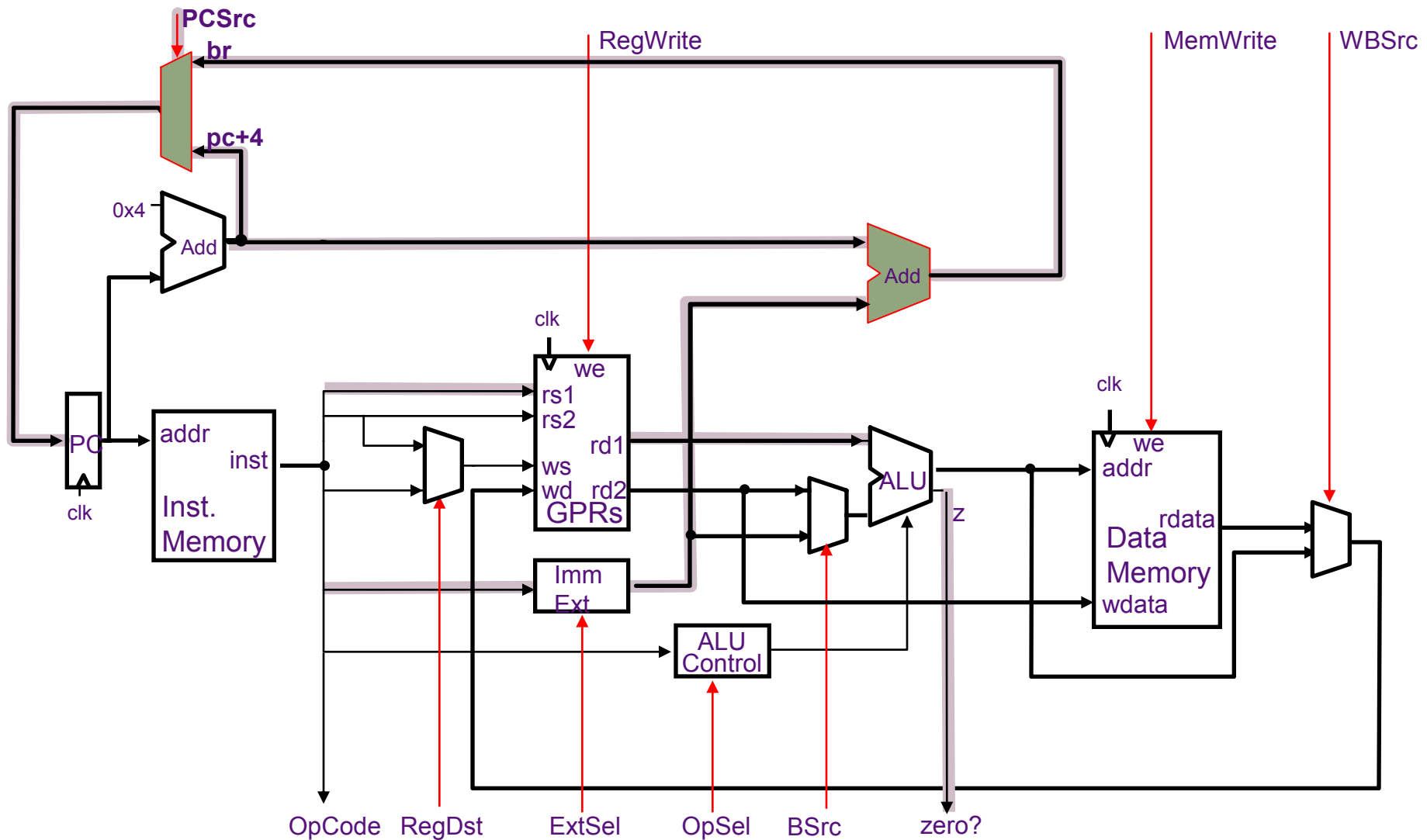
Conditional (on GPR) PC-relative branch

6	5	5	16
opcode	rs		offset

BEQZ, BNEZ

Unconditional register-indirect jumps

6	5	5	16
opcode	rs		

JR, JALR
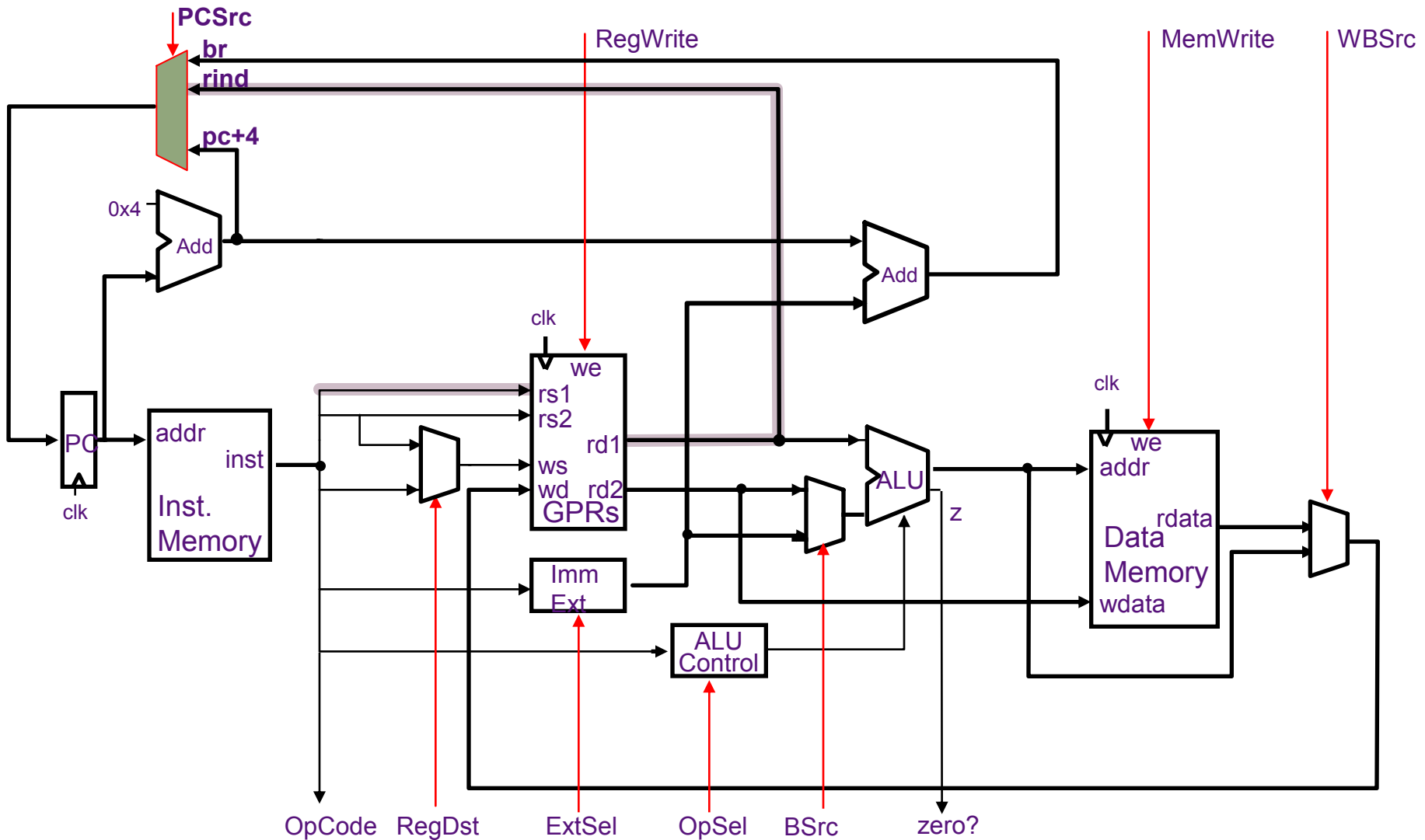
Unconditional absolute jumps

6	26
opcode	target

J, JAL

- PC-relative branches add offset×4 to PC+4 to calculate the target address (offset is in words): ±128 KB range
- Absolute jumps append target×4 to PC<31:28> to calculate the target address: 256 MB range
- 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)

PCSrc

br

rind

pc+4

RegWrite

MemWrite

WBSrc

0x4

Add

Add

clk

we

rs1

rs2

31

rd1

ws

wd   rd2

GPRs

clk

we

addr

rdata

Data
Memory

wdata

PC

clk

addr

inst

Inst.
Memory

Imm
Ext

ALU

z

ALU
Control

OpCode   **RegDst**   ExtSel   OpSel   BSrc   zero?
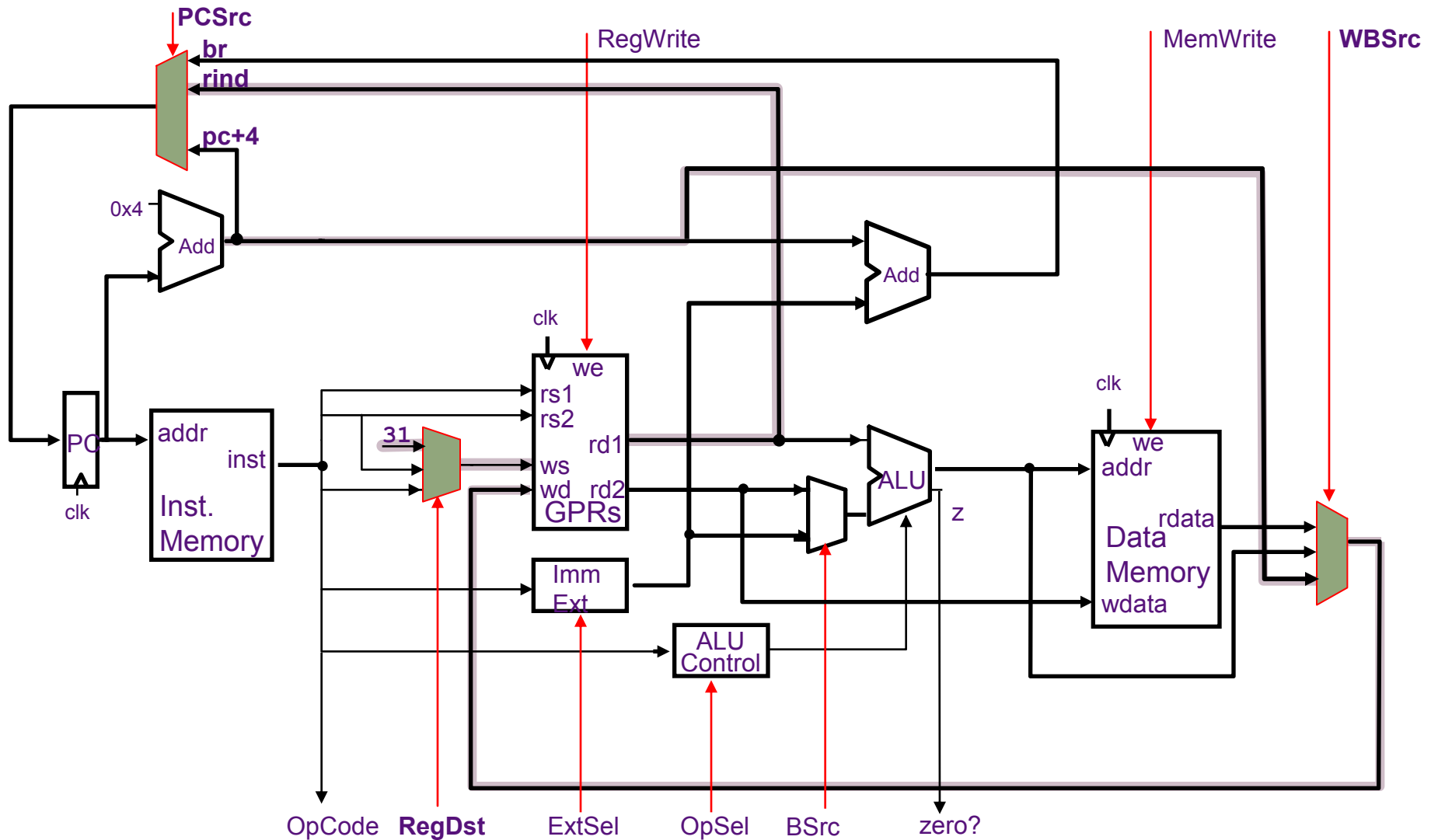
Absolute Jumps (J, JAL)
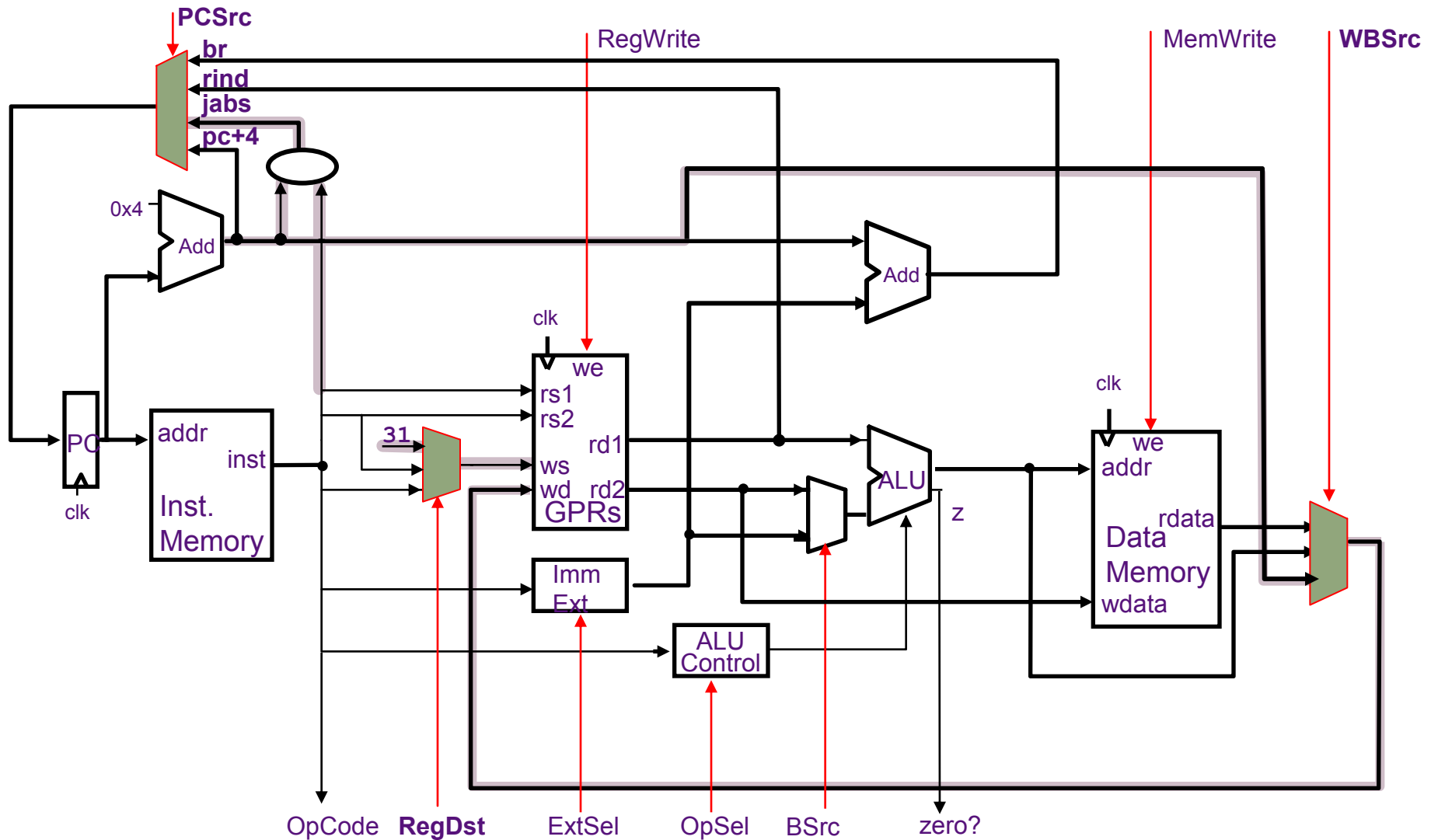
Harvard-Style Datapath for MIPS
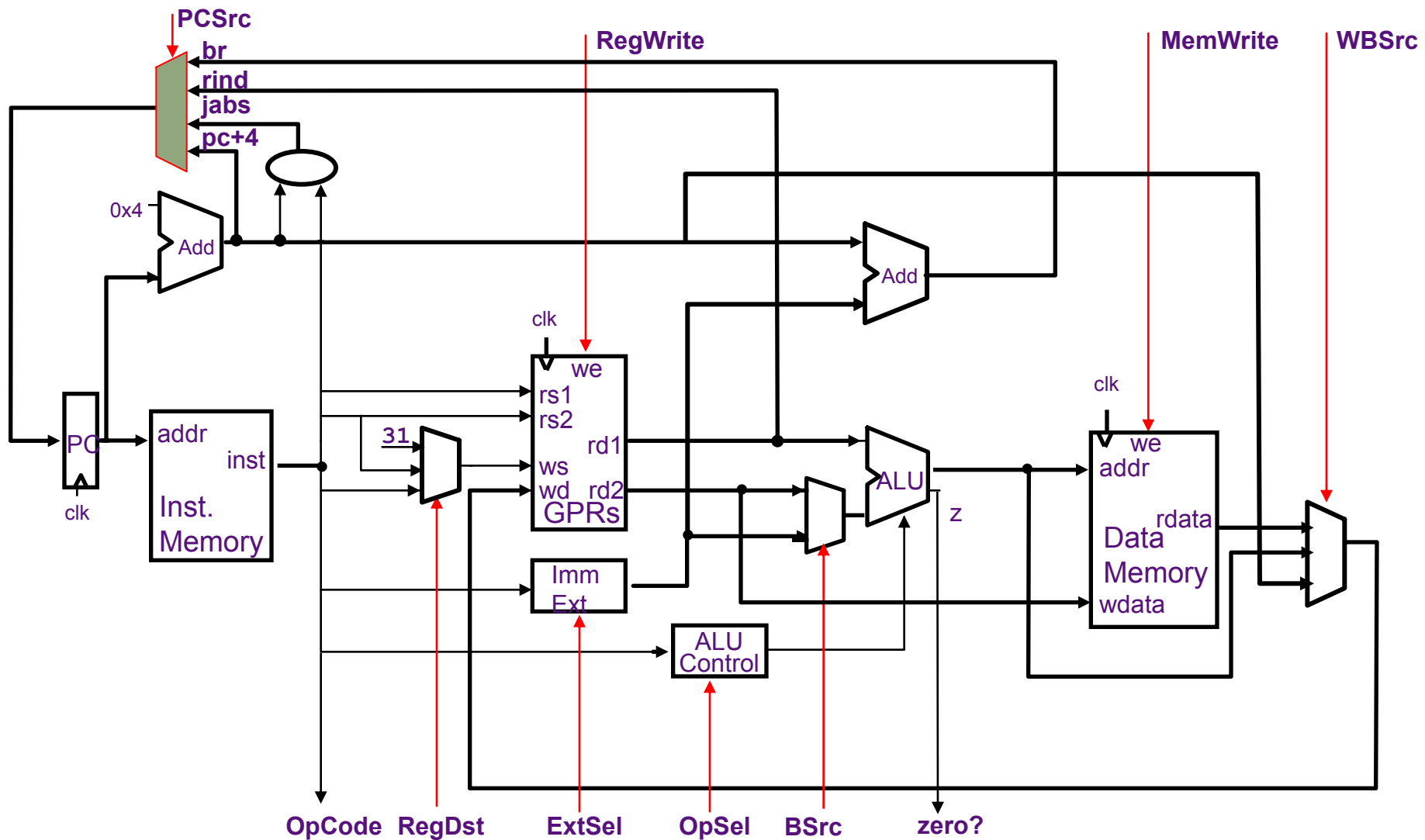
Hardwired Control is pure Combinational Logic

op code →

zero? →

combinational logic

→ ExtSel

→ BSrc

→ OpSel

→ MemWrite

→ WBSrc

→ RegDst

→ RegWrite
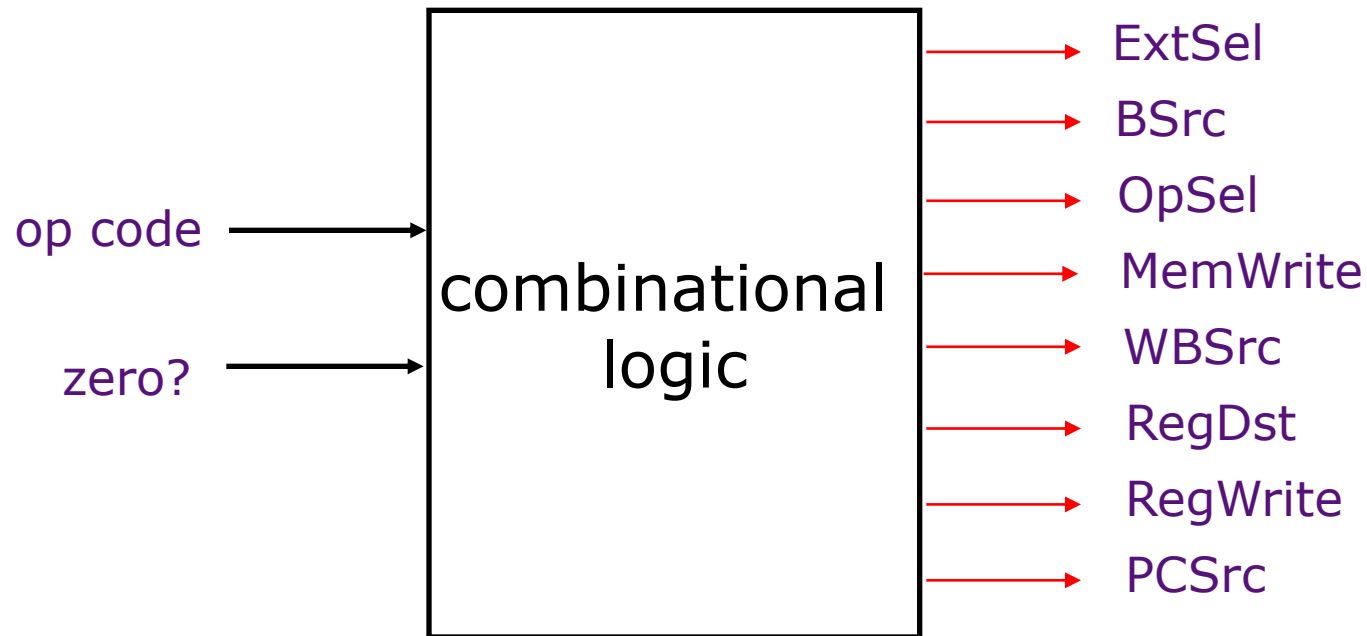
→ PCSrc

ALU Control & Immediate Extension

Inst<5:0> *(Func)*

Inst<31:26> *(Opcode)*

+

0?

ALUop

OpSel
(Func, Op, +, 0?)

Decode Map

ExtSel
($sExt_{16}$, $uExt_{16}$, $High_{16}$)
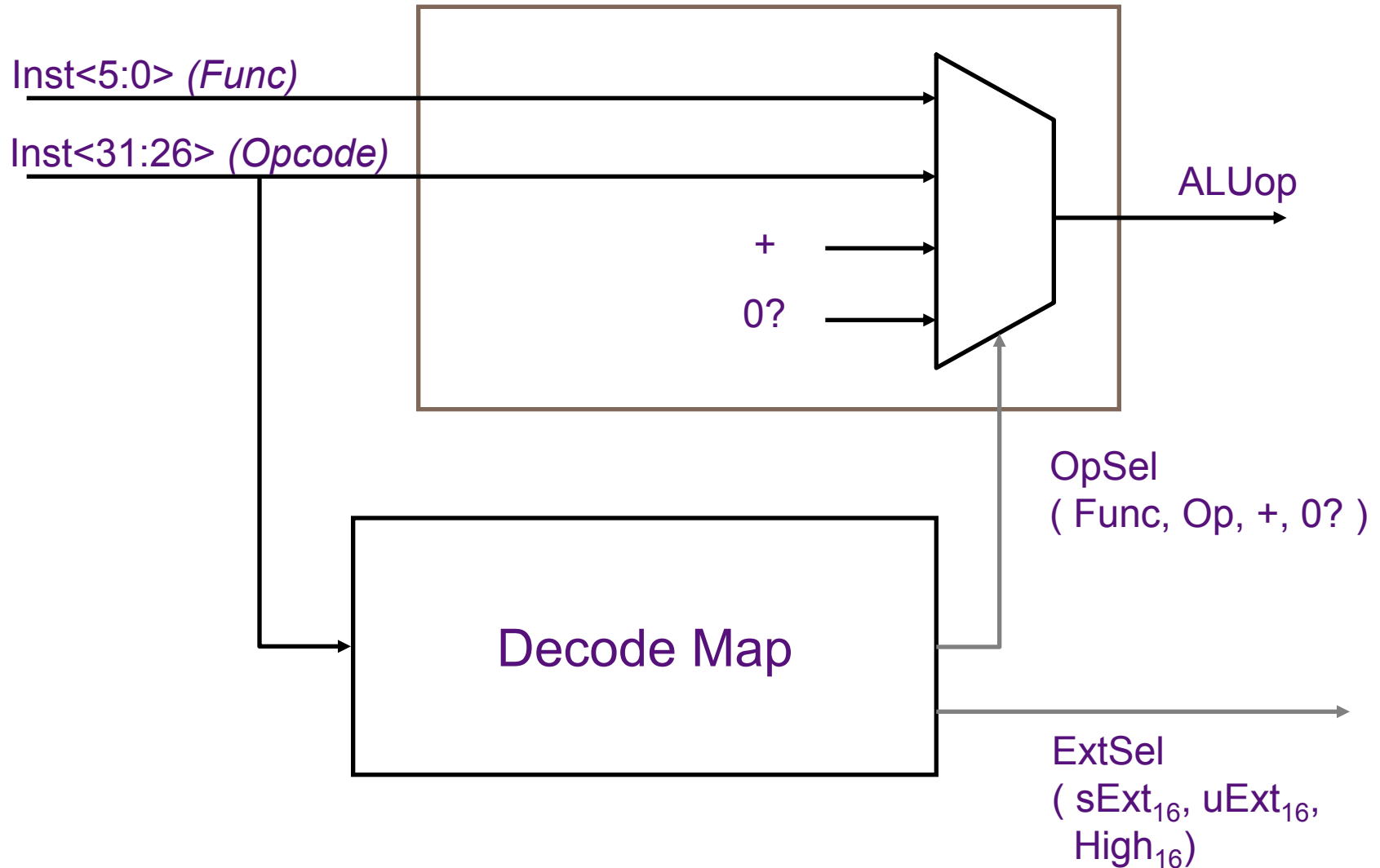
Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	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
$BEQZ_{z=0}$	$sExt_{16}$	*	0?	no	no	*	*	br
$BEQZ_{z=1}$	$sExt_{16}$	*	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   WBSrc = ALU / Mem / PC
RegDst = rt / rd / R31   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
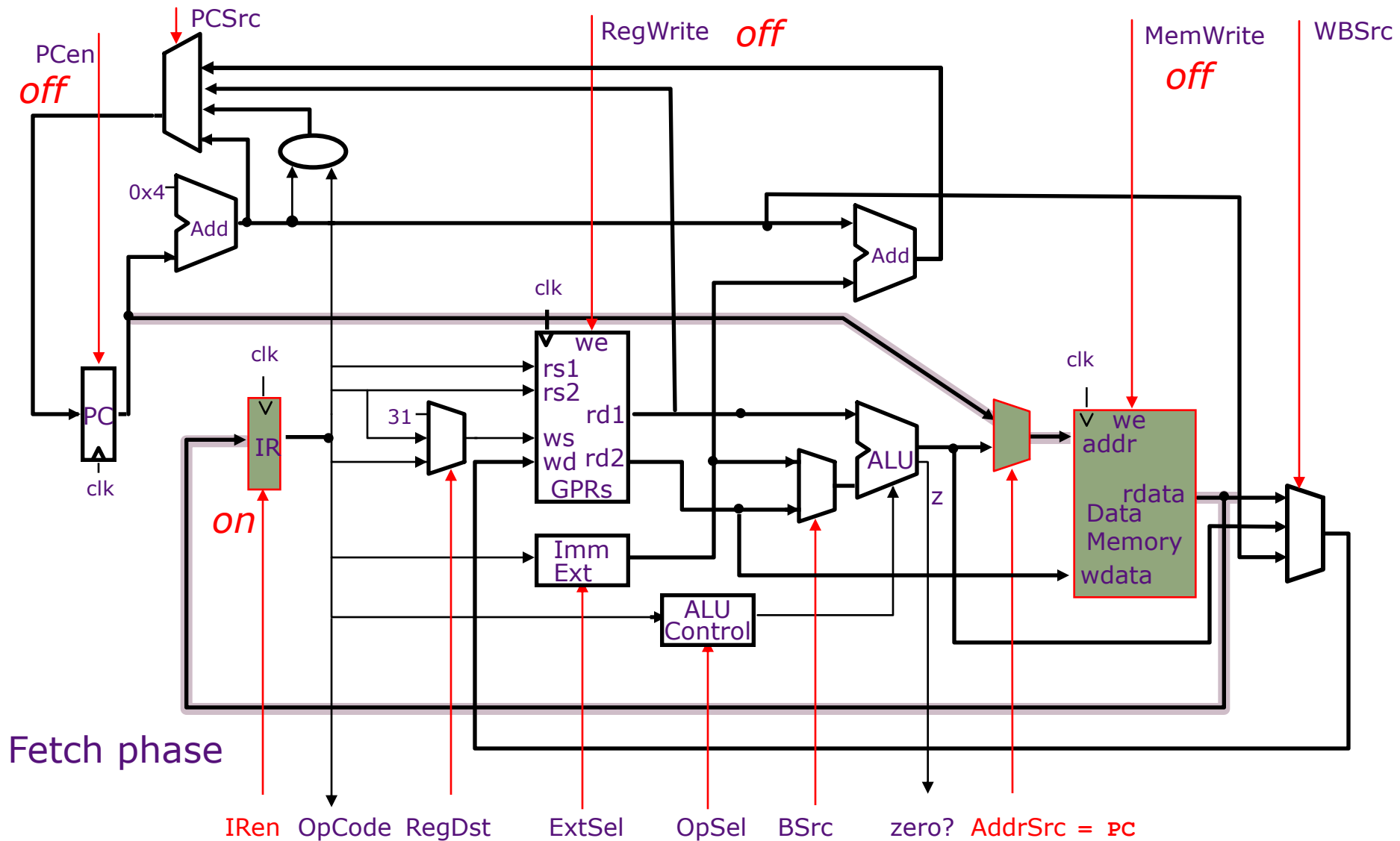
Princeton challenge

- 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
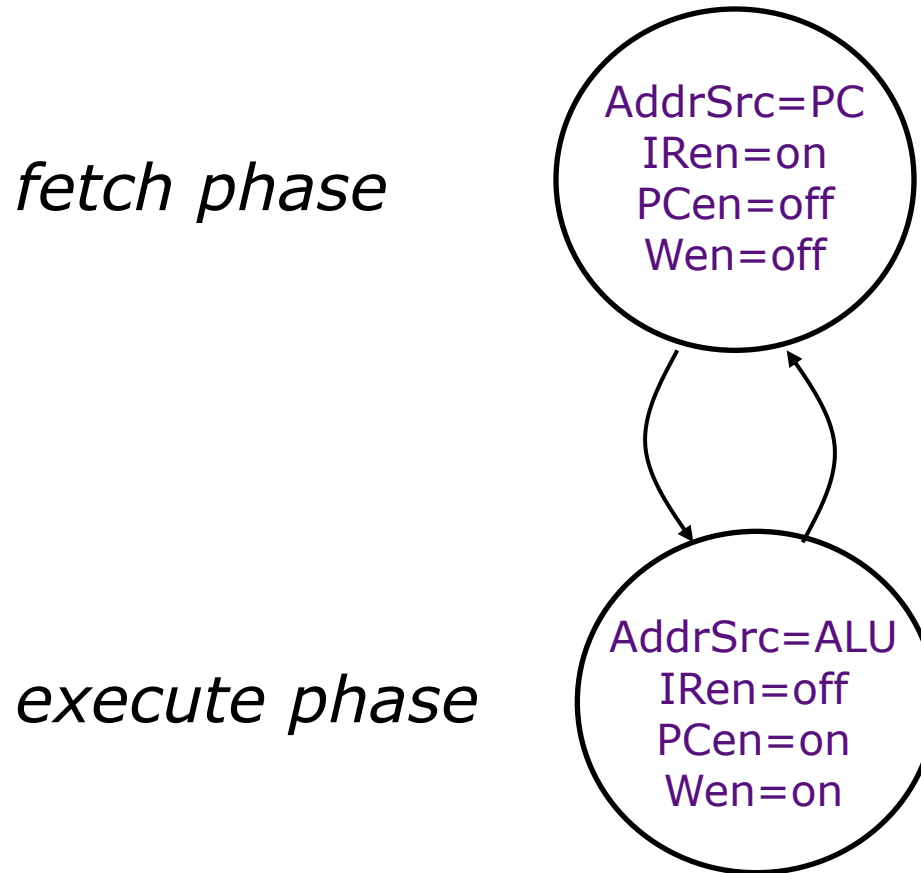
Structural hazard

Princeton Microarchitecture
Datapath & Control

Fetch phase

PCSrc PCen *off* RegWrite *off* MemWrite *off* WBSrc

0x4 Add Add

clk we rs1 rs2 rd1 ws wd rd2 GPRs

31 clk IR *on*

Imm Ext

ALU Control

clk ALU z

clk we addr rdata Data Memory wdata

PC clk

IRen OpCode RegDst ExtSel OpSel BSrc zero? AddrSrc = PC

Two-State Controller:
Princeton Architecture

fetch phase

AddrSrc=PC
IRen=on
PCen=off
Wen=off

execute phase

AddrSrc=ALU
IRen=off
PCen=on
Wen=on

A flipflop can be used to remember the phase

Hardwired Controller:
Princeton Architecture

IR

op code

zero?

old combinational logic (Harvard)

ExtSel, BSrc, OpSel, WBSrc, RegDest, PCsrc1, PCsrc2

MemWrite

RegWrite

S

new combinational logic

Wen

PCen

IRen

AddrSrc

1-bit Toggle FF
I-fetch / Execute
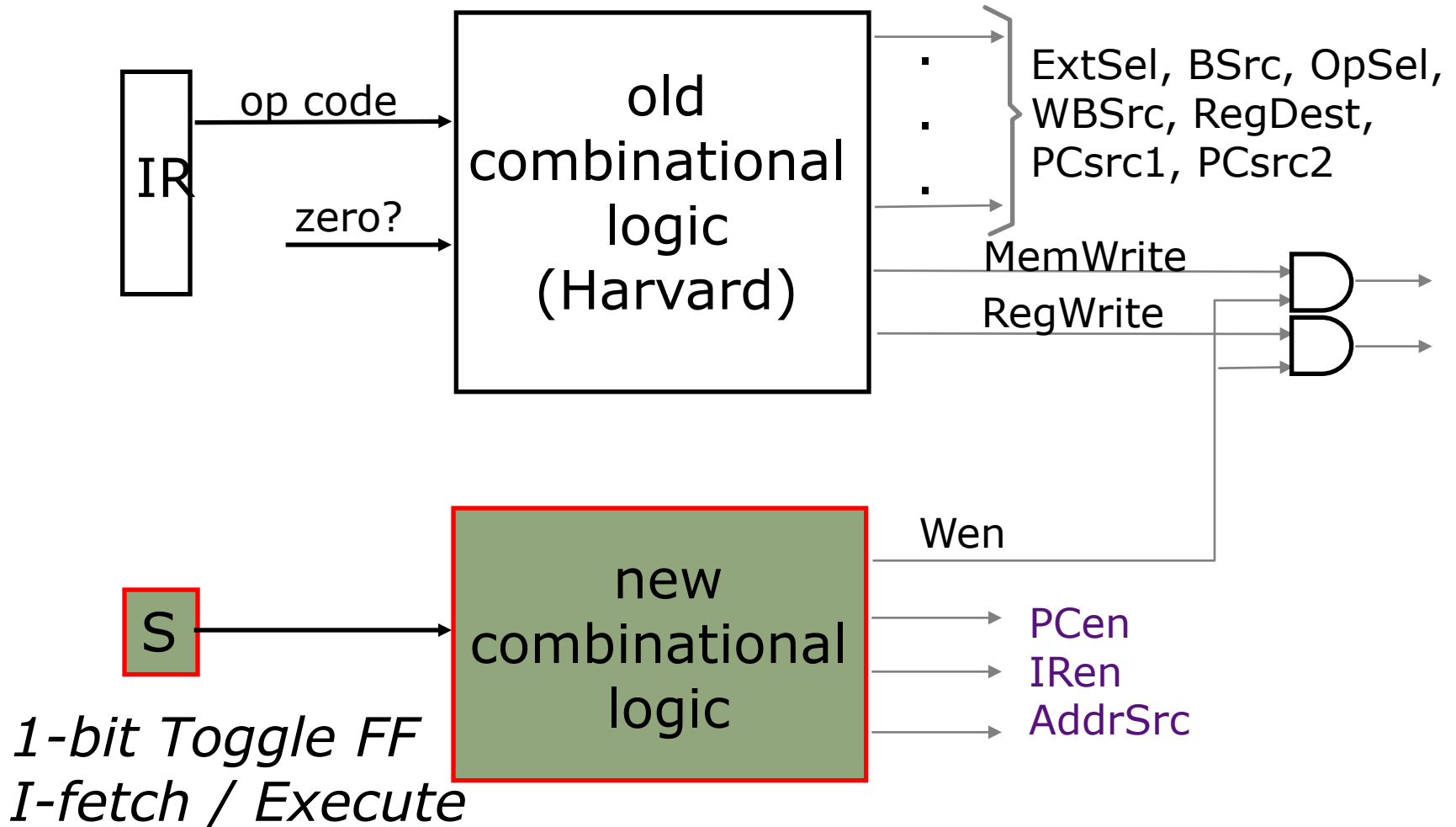
Clock Rate vs CPI

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

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

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

Suppose $t_M >> t_{RF} + t_{ALU} + t_{WB}$

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

$CPI_{Princeton} = 2$

$CPI_{Harvard} = 1$

No difference in performance!

Is it possible to design a controller for the Princeton architecture with CPI < 2 ?

CPI = Clock cycles Per Instruction

Stay tuned!