Hardwired, Non-pipelined ISA Implementation

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

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*

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

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

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

this lecture $\longrightarrow$

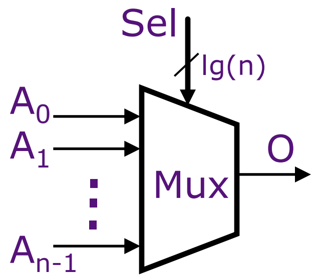
Microarchitecture	CPI	cycle time
Microcoded	>1	short
Single-cycle unpipelined	1	long
Pipelined	1	short

Hardware Elements

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

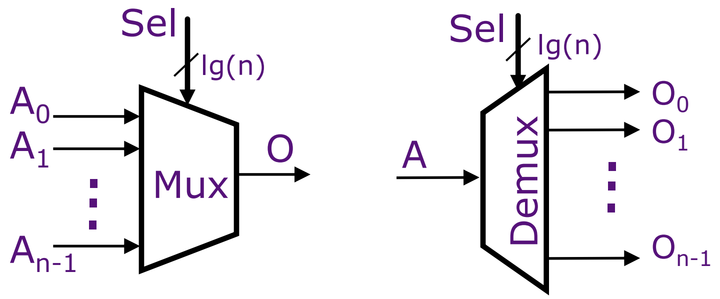
Hardware Elements

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



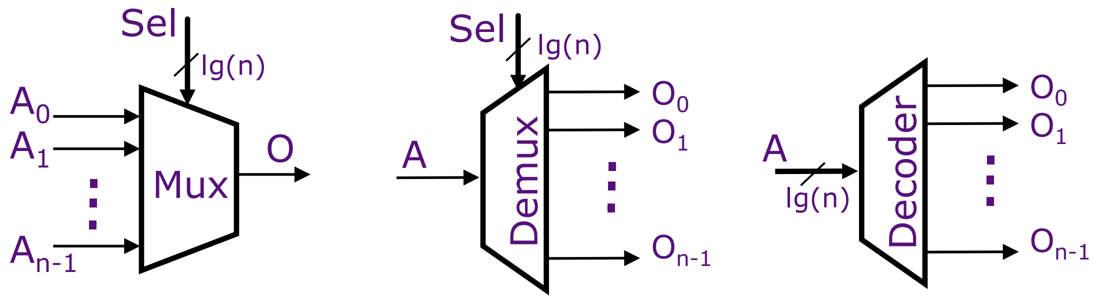
Hardware Elements

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



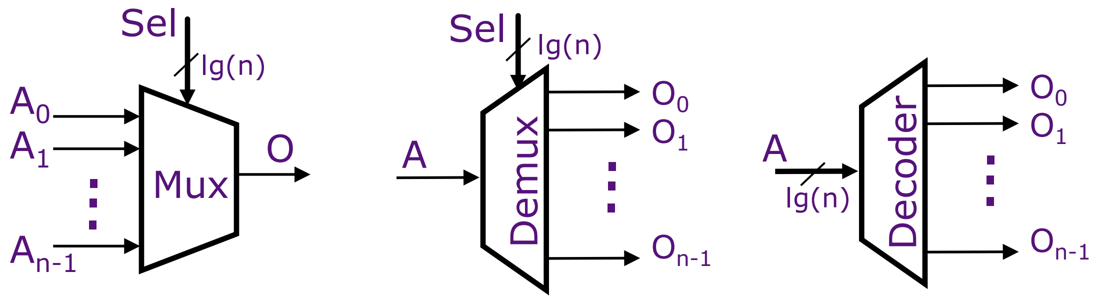
Hardware Elements

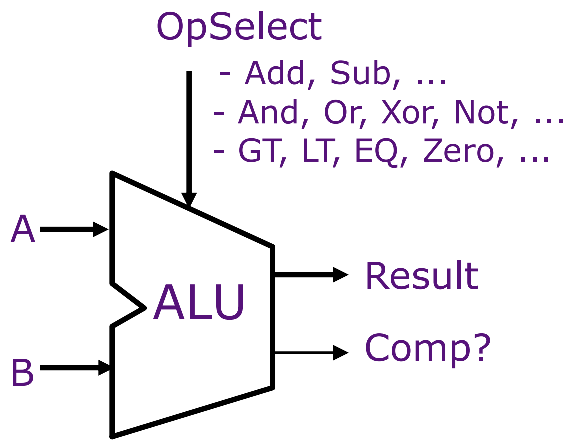
- Combinational circuits
  - Mux, Demux, Decoder, ALU, ...



Hardware Elements

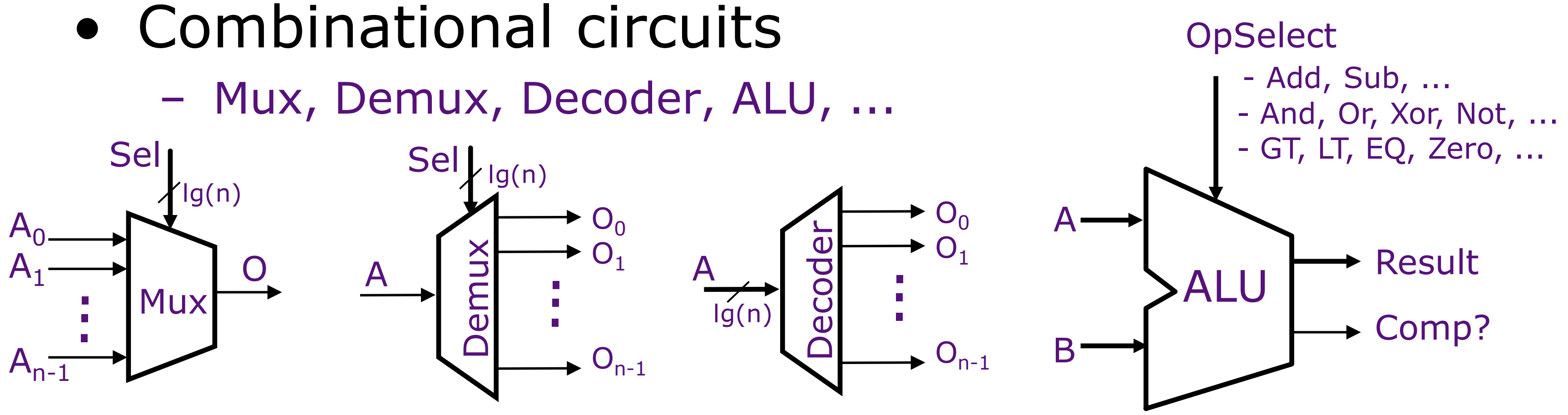
- Combinational circuits
  - Mux, Demux, Decoder, ALU, ...





Hardware Elements

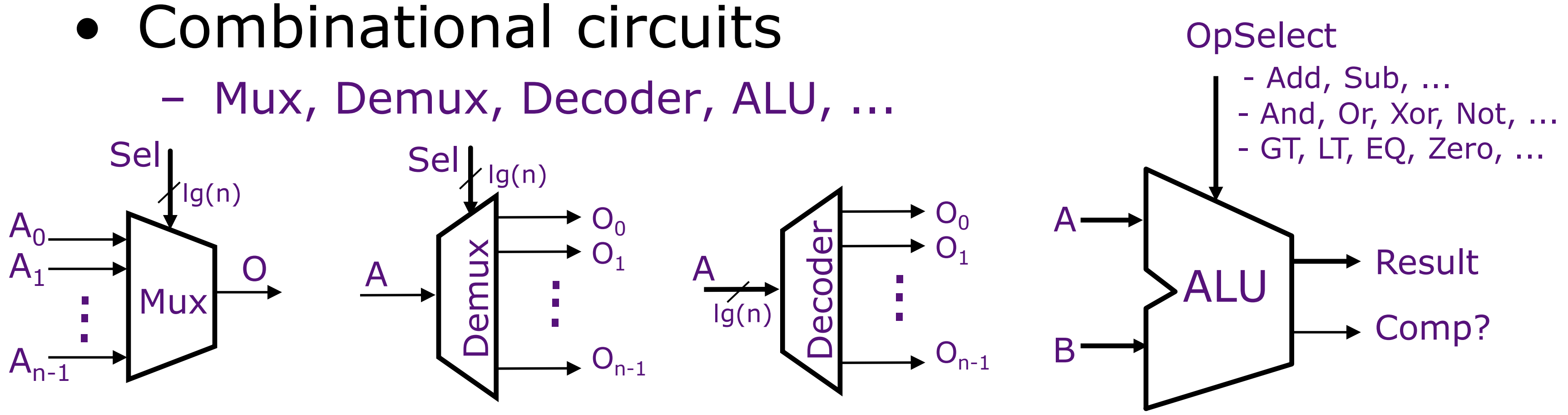
- Combinational circuits
  - Mux, Demux, Decoder, ALU, ...



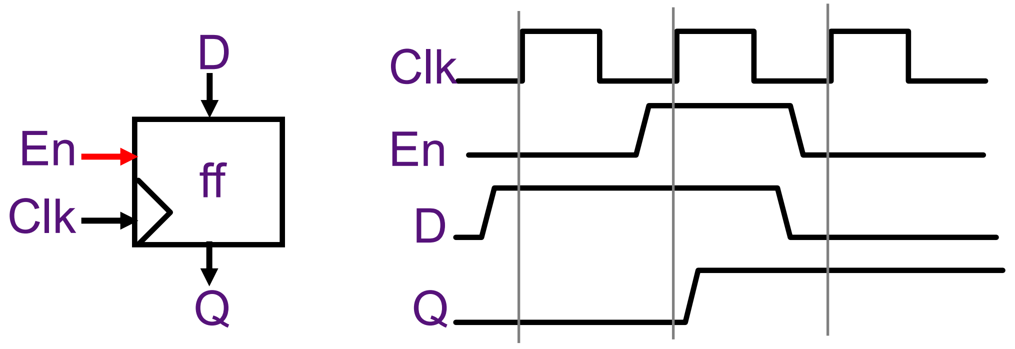
- Synchronous state elements
  - Flipflop, Register, Register file, SRAM, DRAM

Hardware Elements

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



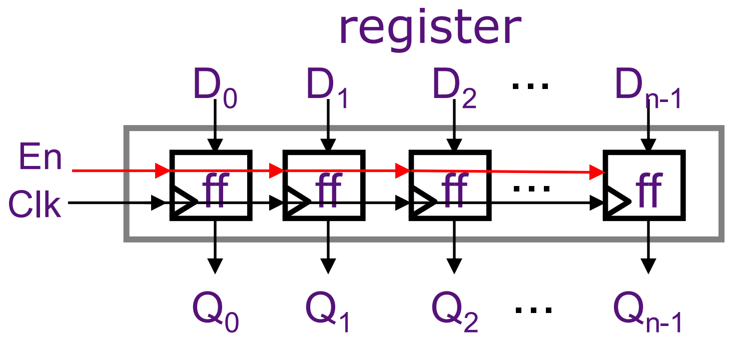
- Synchronous state elements
  - Flipflop, Register, Register file, SRAM, DRAM



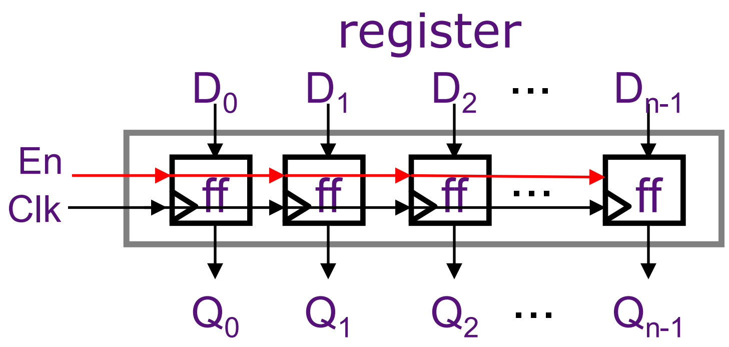
Edge-triggered: Data is sampled at the rising edge

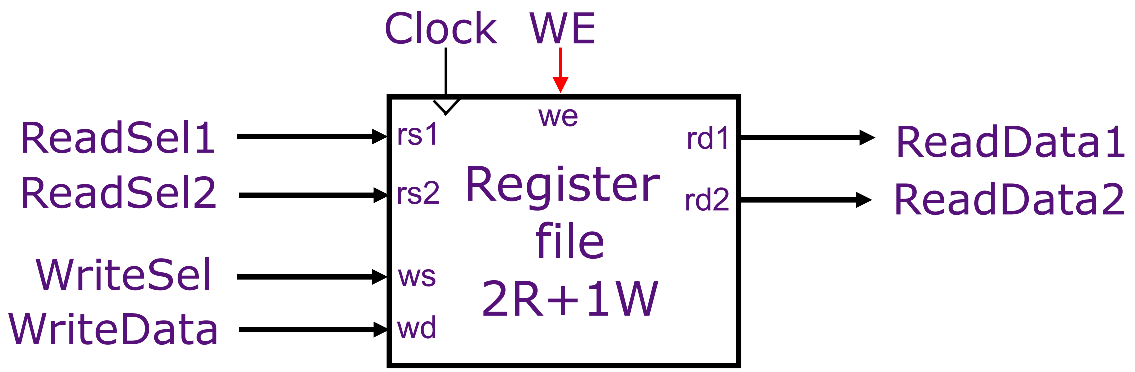
Register Files

Register Files

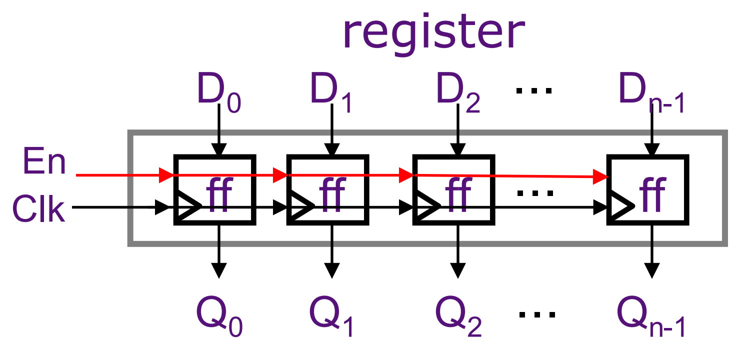


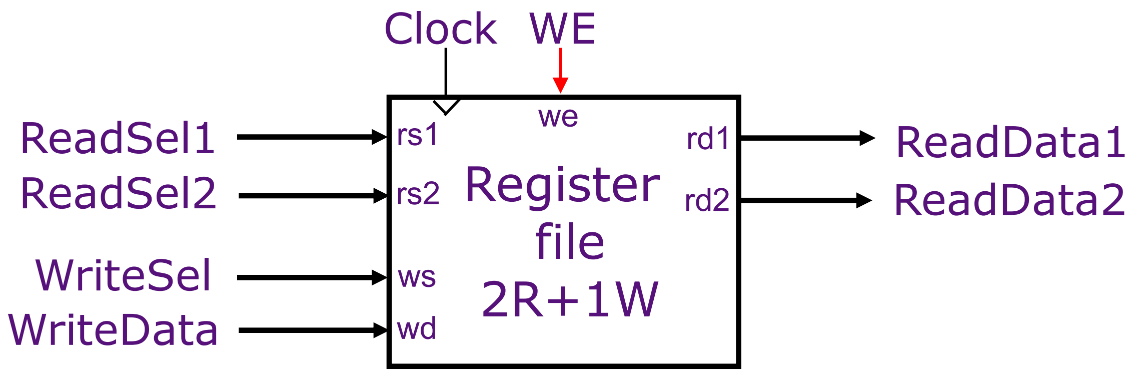
Register Files





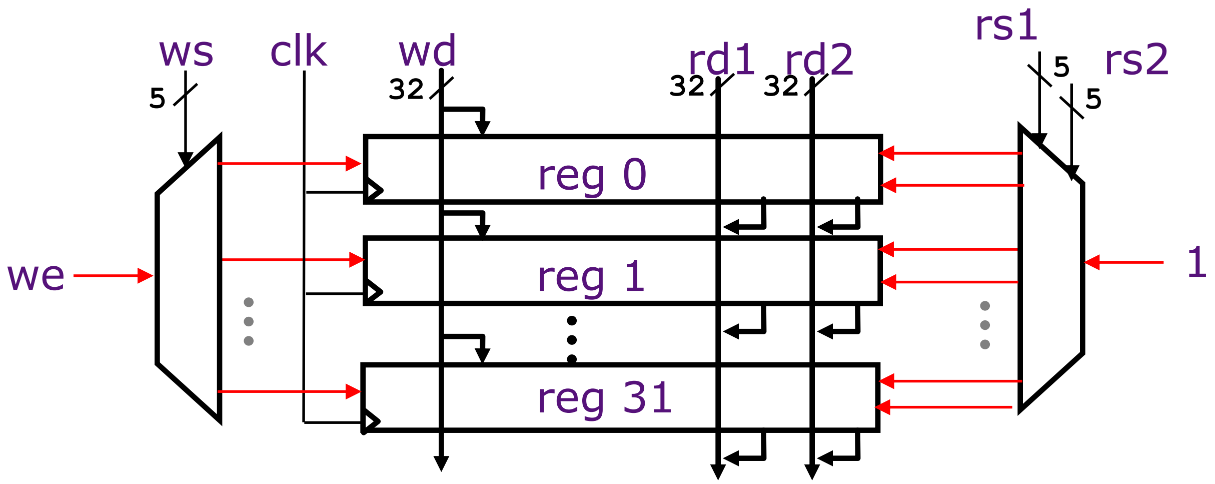
Register Files





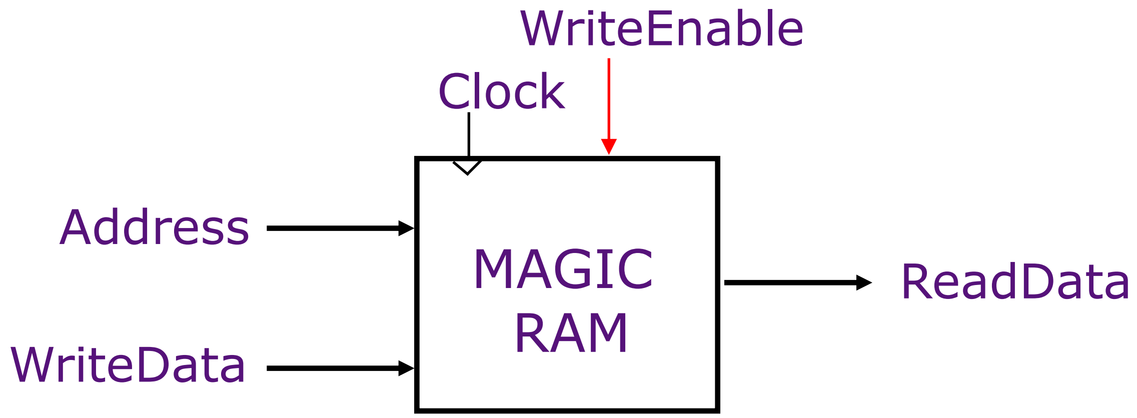
No timing issues in reading a selected register

Register File Implementation



- Register files with a large number of ports are difficult to design
  - 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!!!*

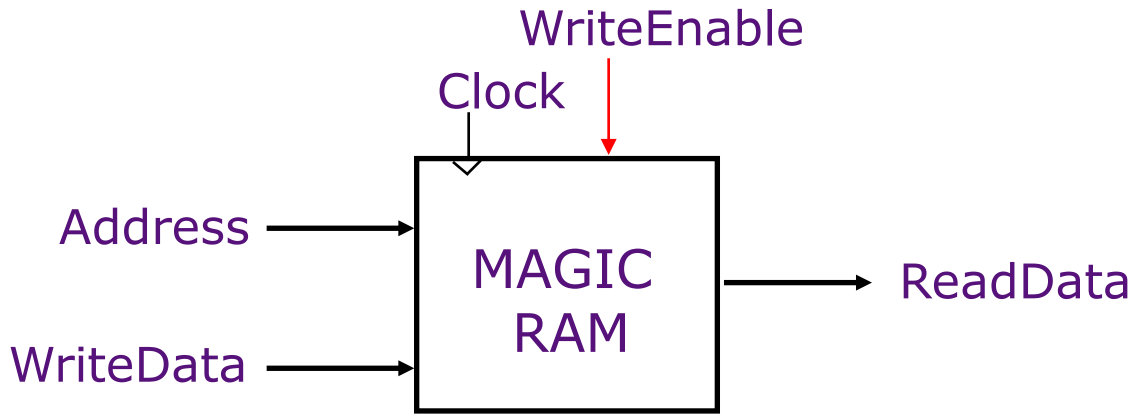
A Simple Memory Model



- 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)

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

Instruction Execution

Execution of an instruction involves

1. Instruction fetch

Instruction Execution

Execution of an instruction involves

1. Instruction fetch
2. Decode

Instruction Execution

Execution of an instruction involves

1. Instruction fetch
2. Decode
3. Register fetch

Instruction Execution

Execution of an instruction involves

1. Instruction fetch
2. Decode
3. Register fetch
4. ALU operation

Instruction Execution

Execution of an instruction involves

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

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

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

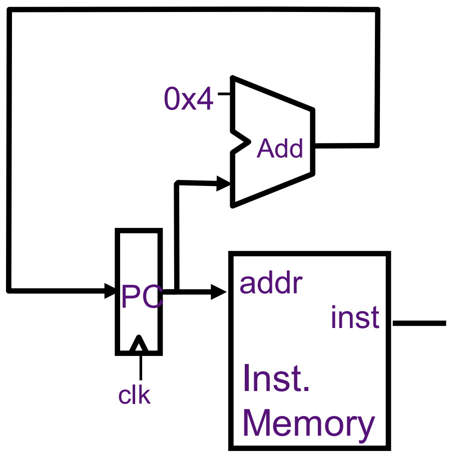
And computing the address of the *next instruction (next PC)*

Datapath: Reg-Reg ALU Instructions

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)

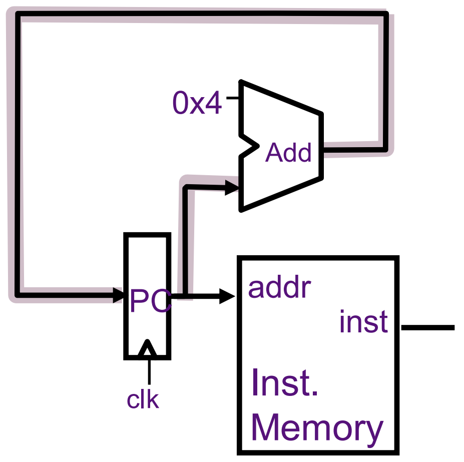
Datapath: Reg-Reg ALU Instructions



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)

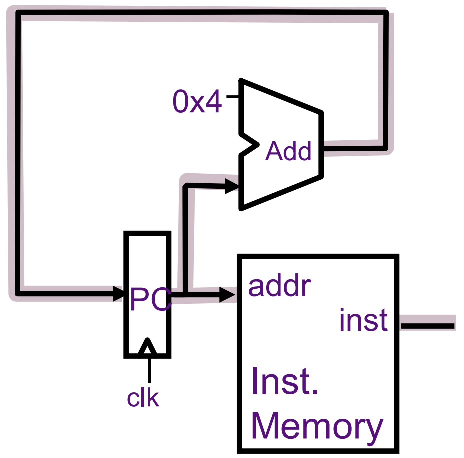
Datapath: Reg-Reg ALU Instructions



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)

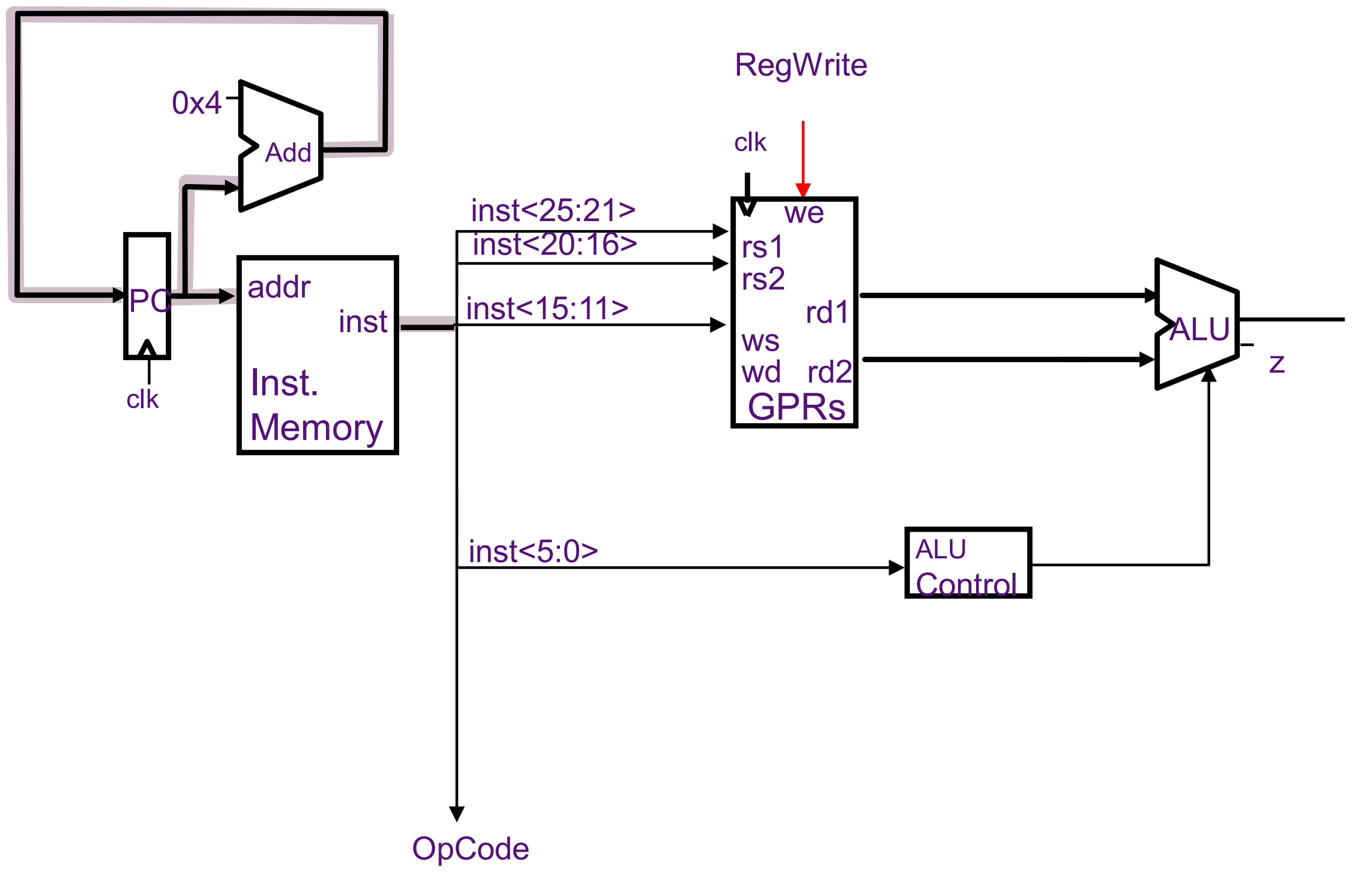
Datapath: Reg-Reg ALU Instructions



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)

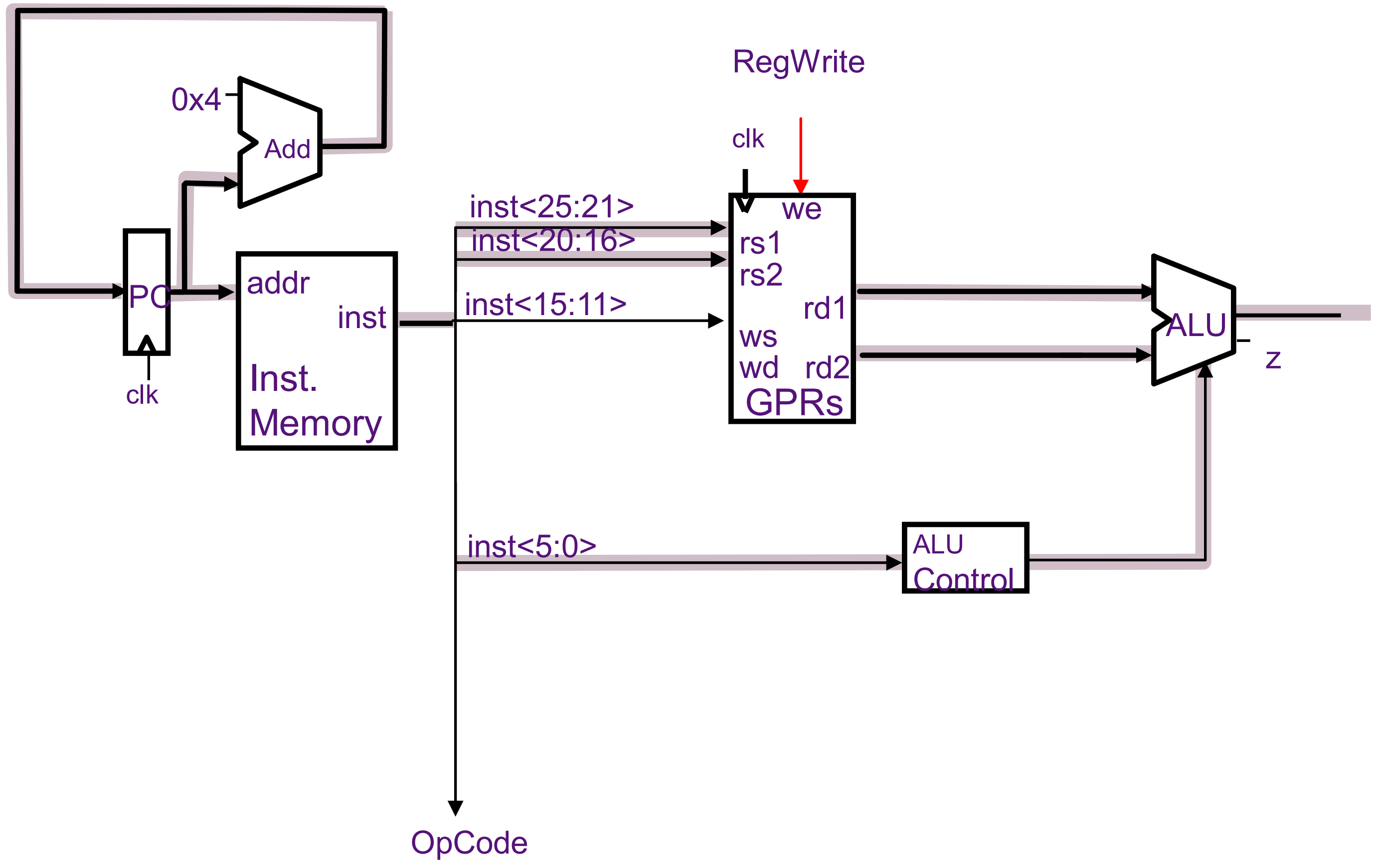
Datapath: Reg-Reg ALU Instructions



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

$rd \leftarrow (rs)\ func\ (rt)$

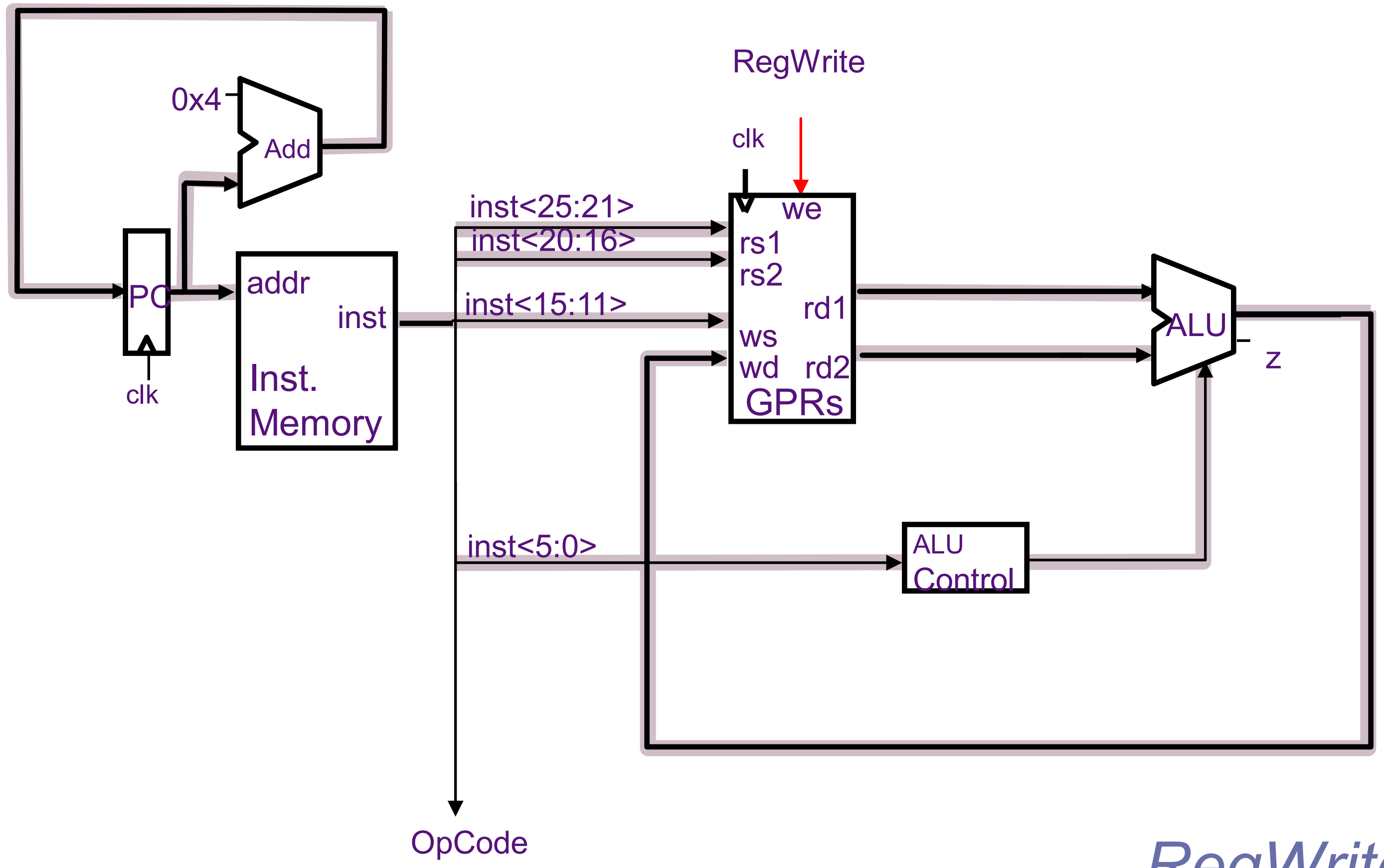
Datapath: Reg-Reg ALU Instructions



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-Reg ALU Instructions

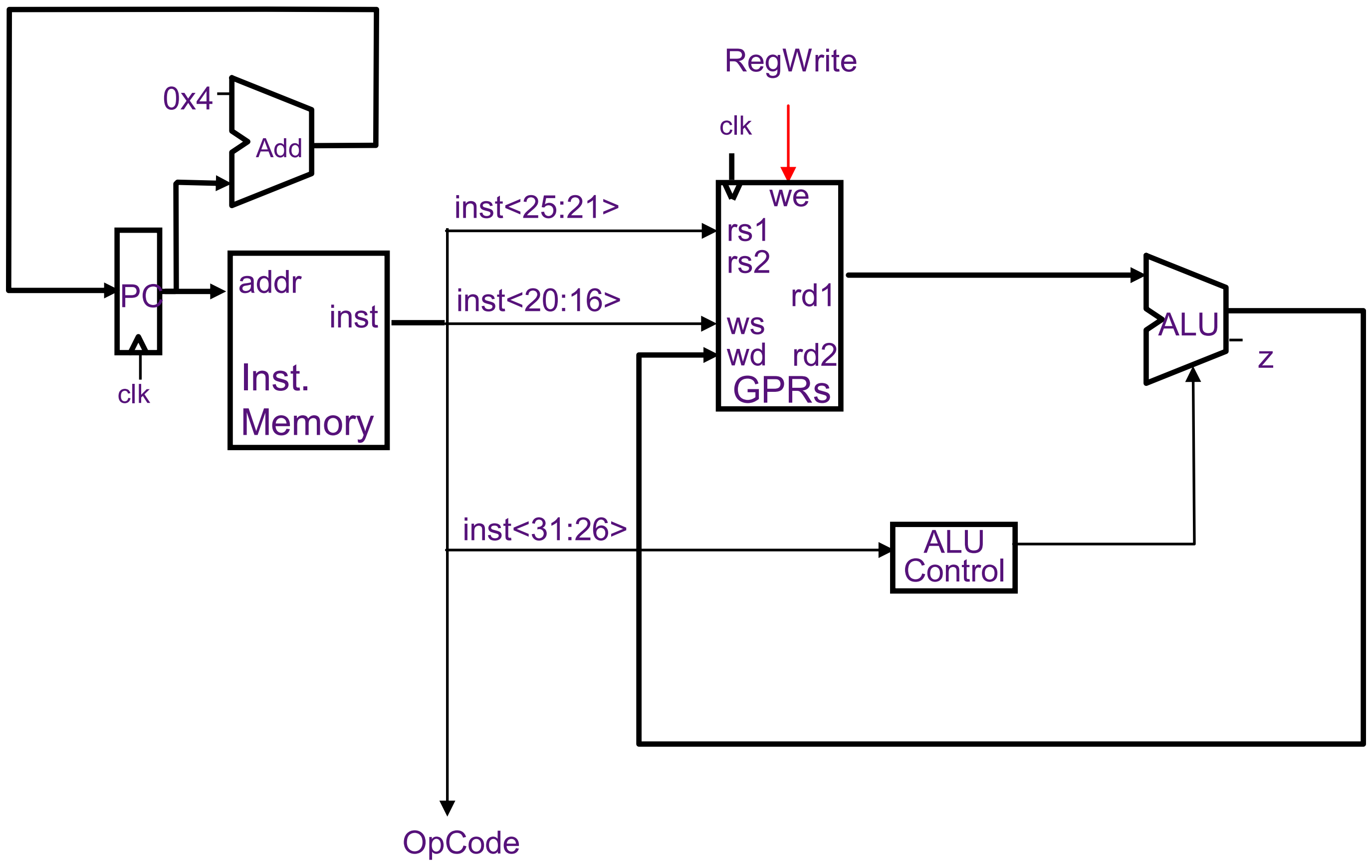


RegWrite Timing?

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)

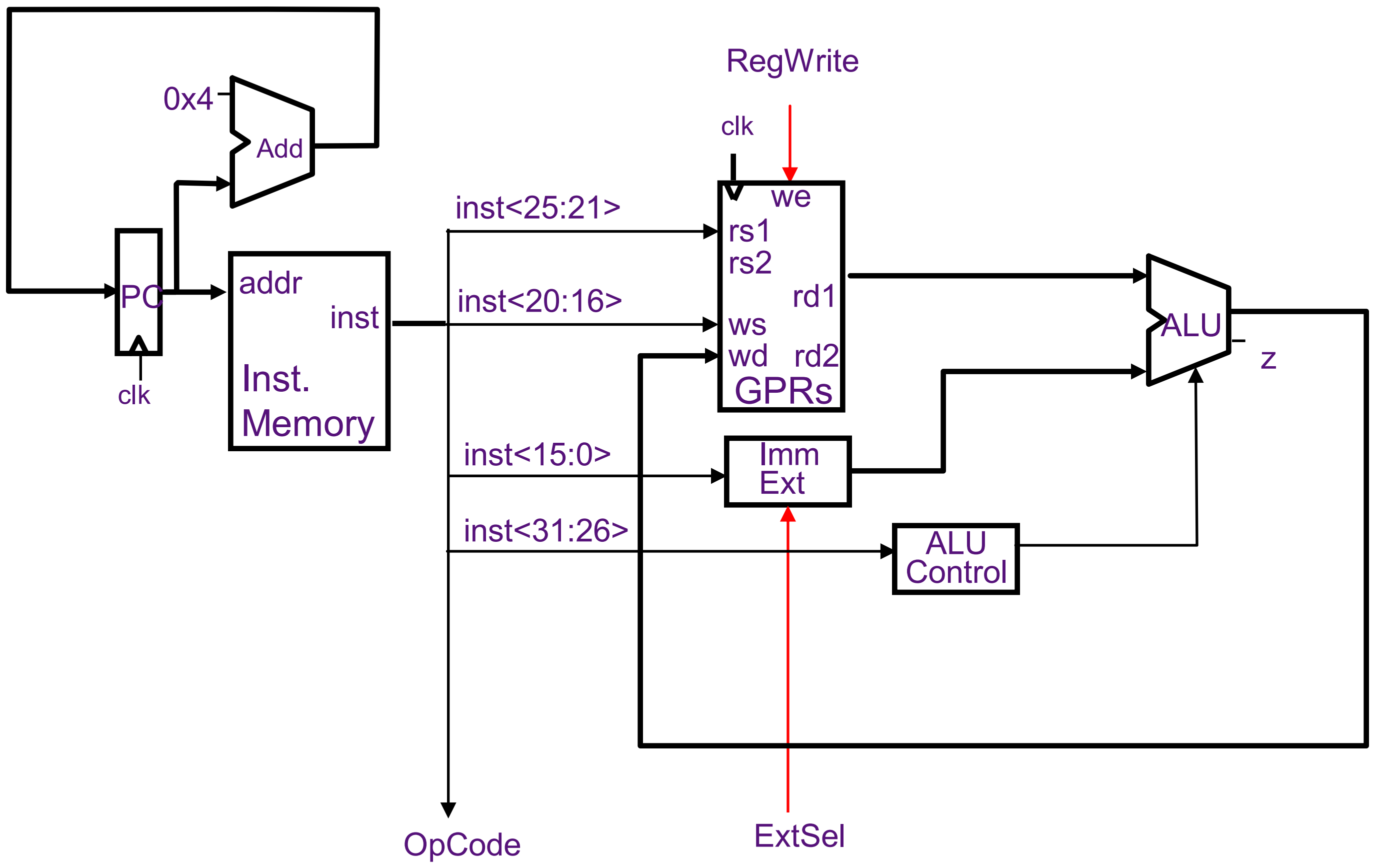
Datapath: Reg-Imm ALU Instructions



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

rt ← (rs) op immediate

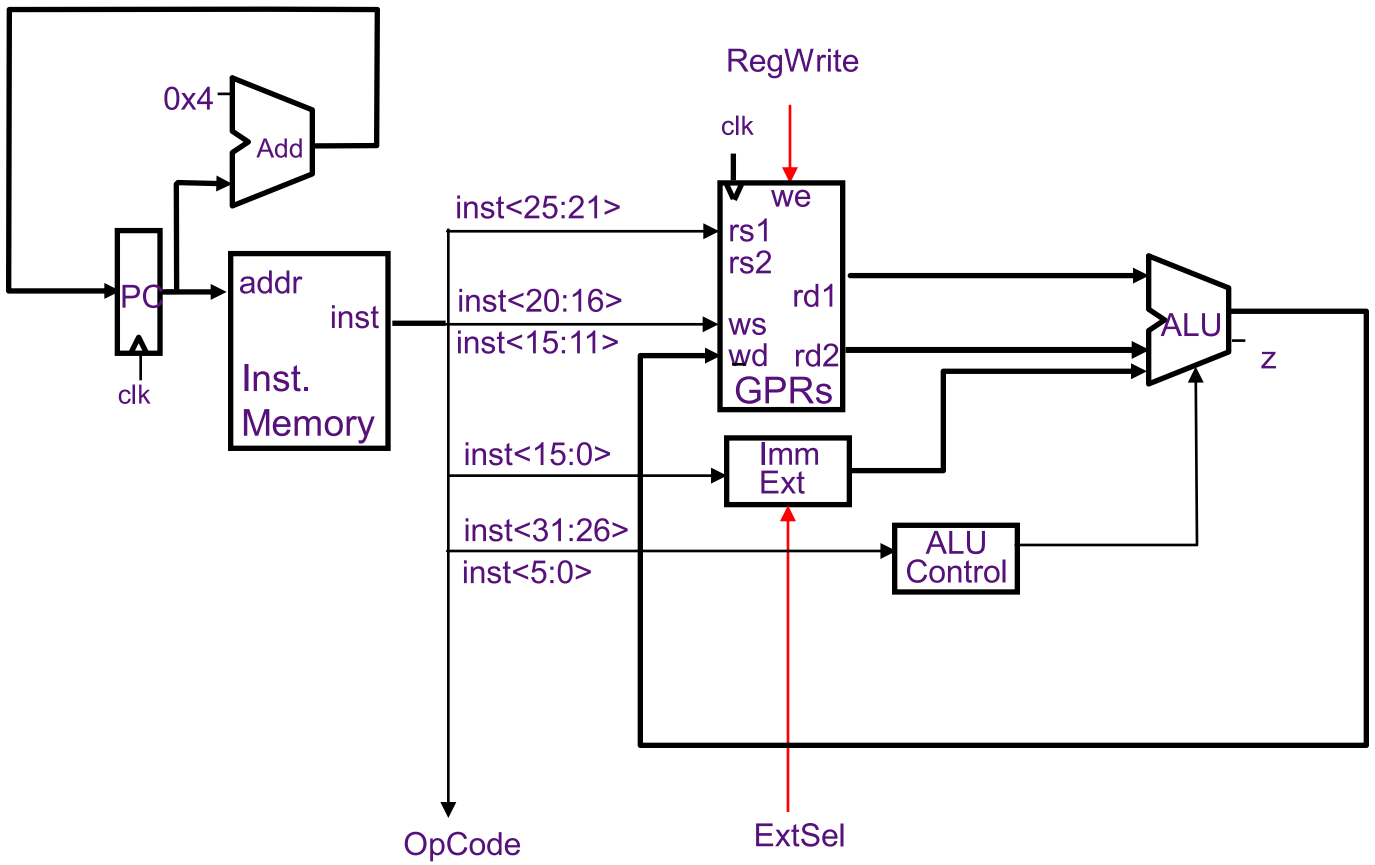
Datapath: Reg-Imm ALU Instructions



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

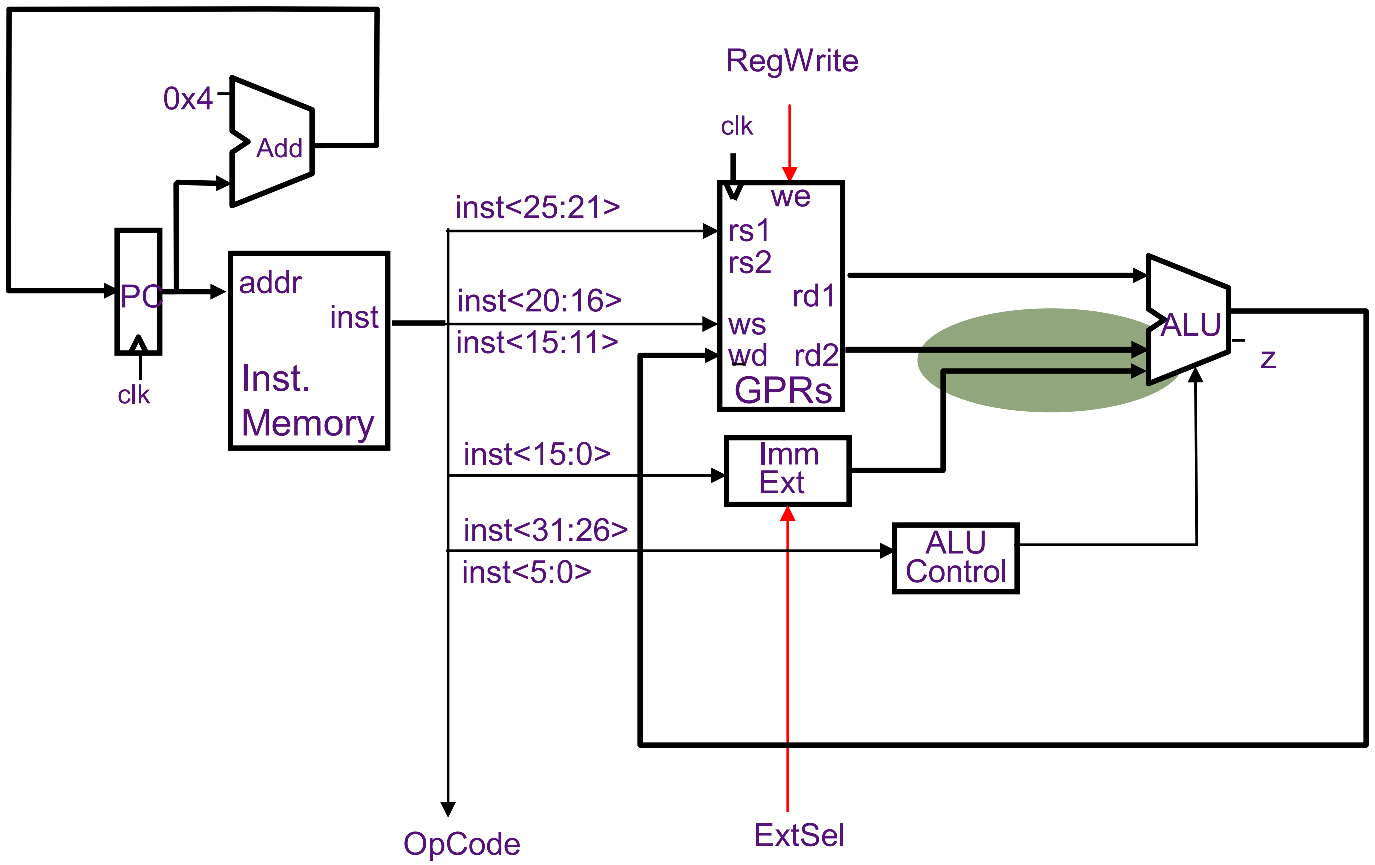
rt ← (rs) op immediate

Conflicts in Merging Datapath



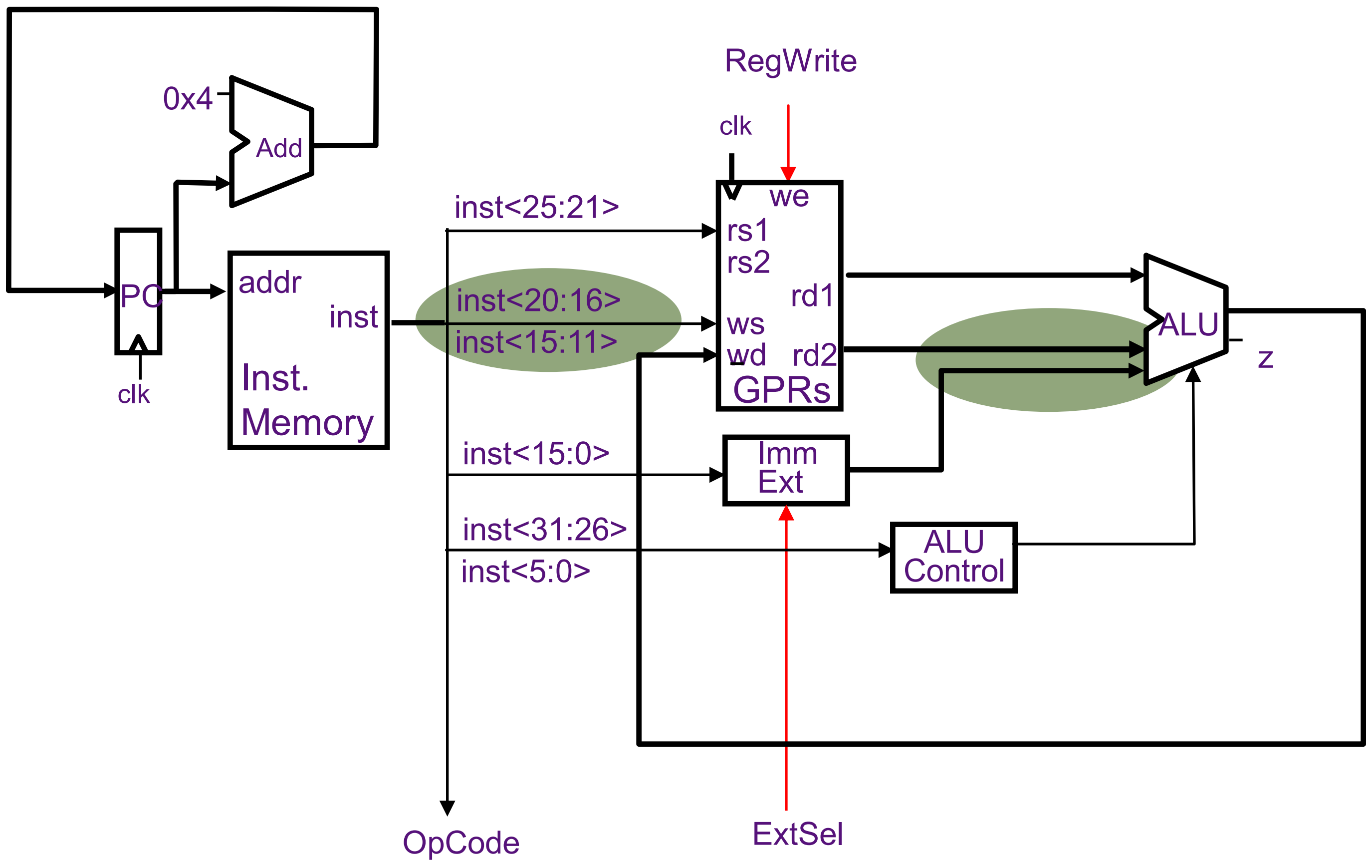
6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Conflicts in Merging Datapath



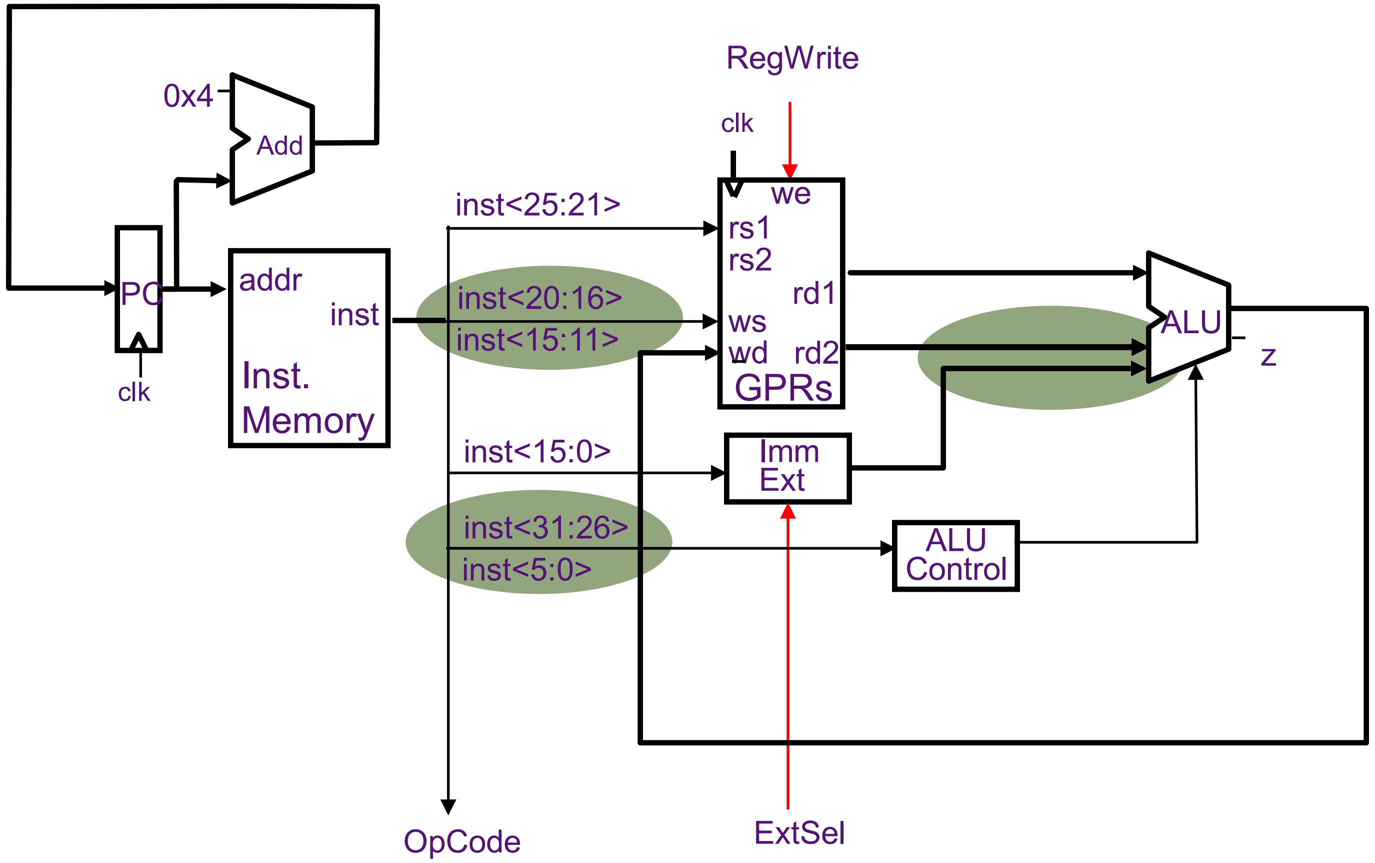
6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Conflicts in Merging Datapath



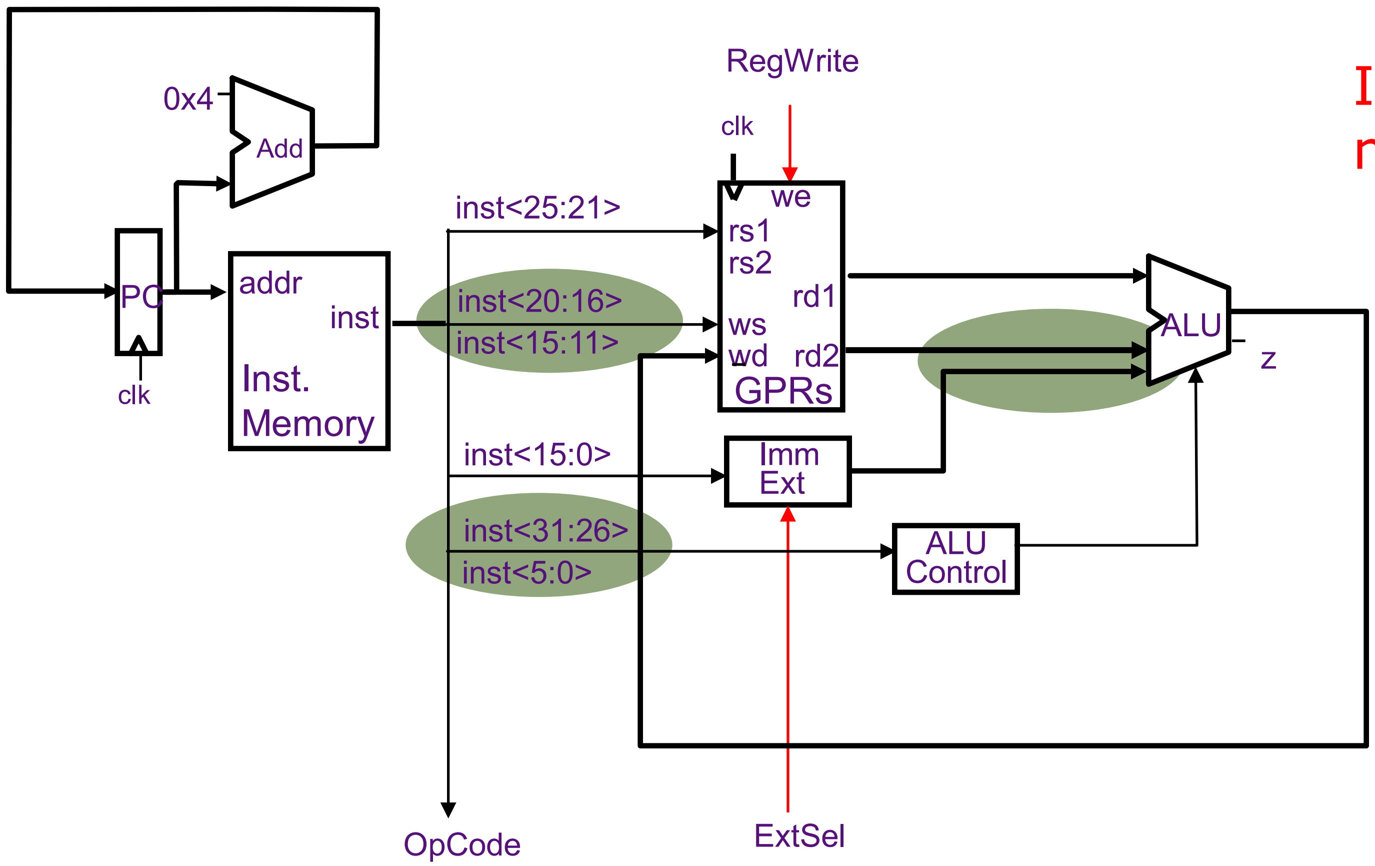
6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Conflicts in Merging Datapath



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

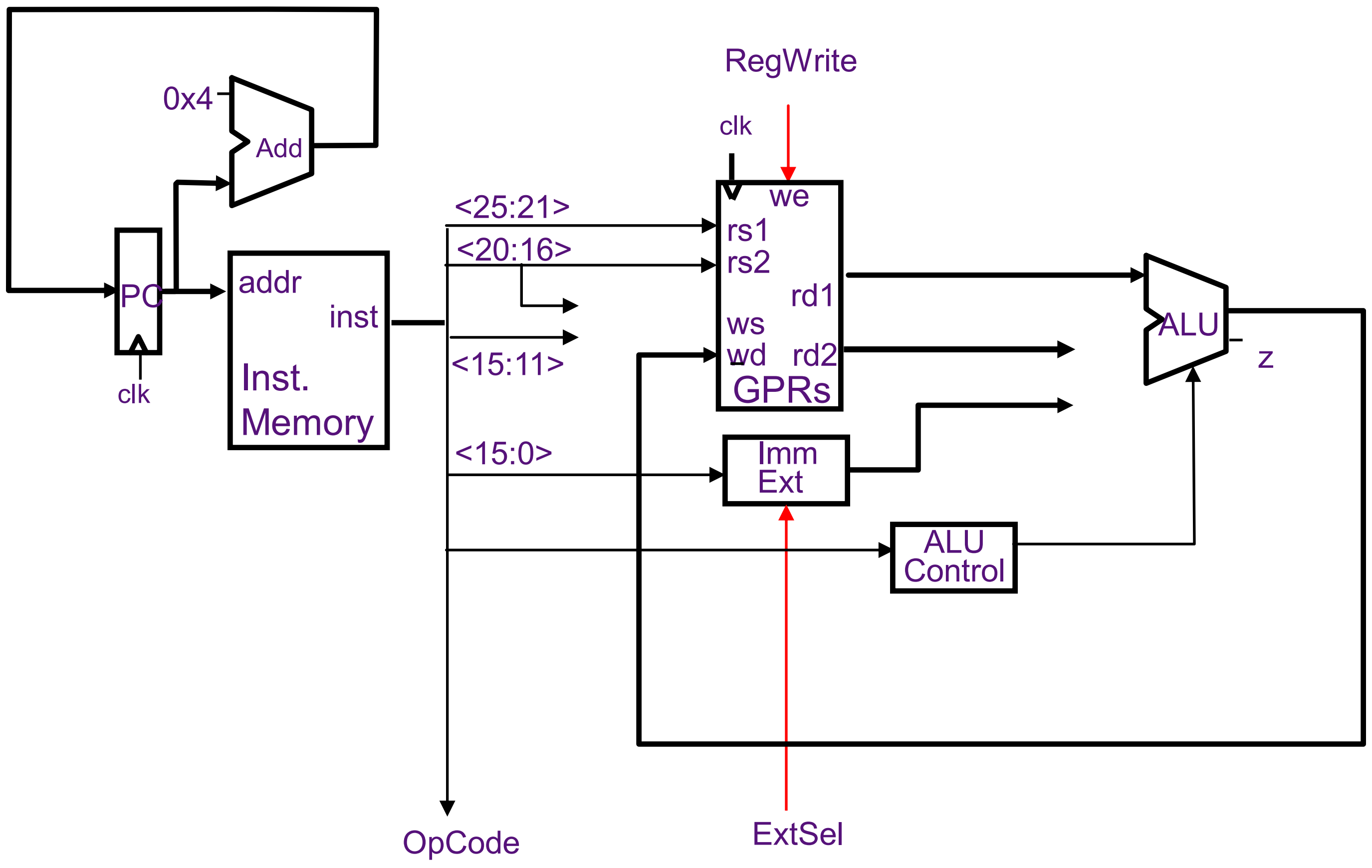
Conflicts in Merging Datapath



Introduce muxes

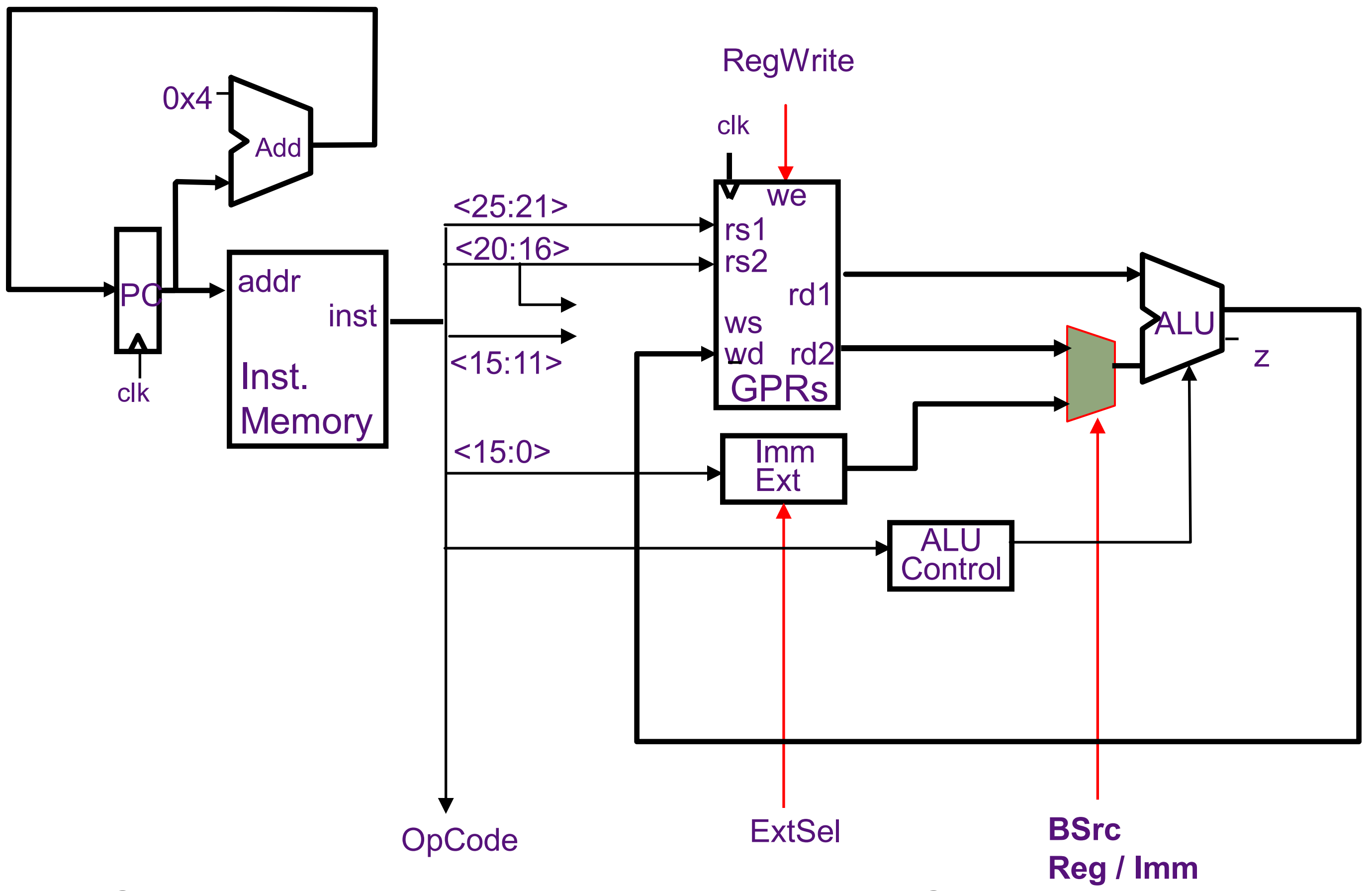
6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Datapath for ALU Instructions



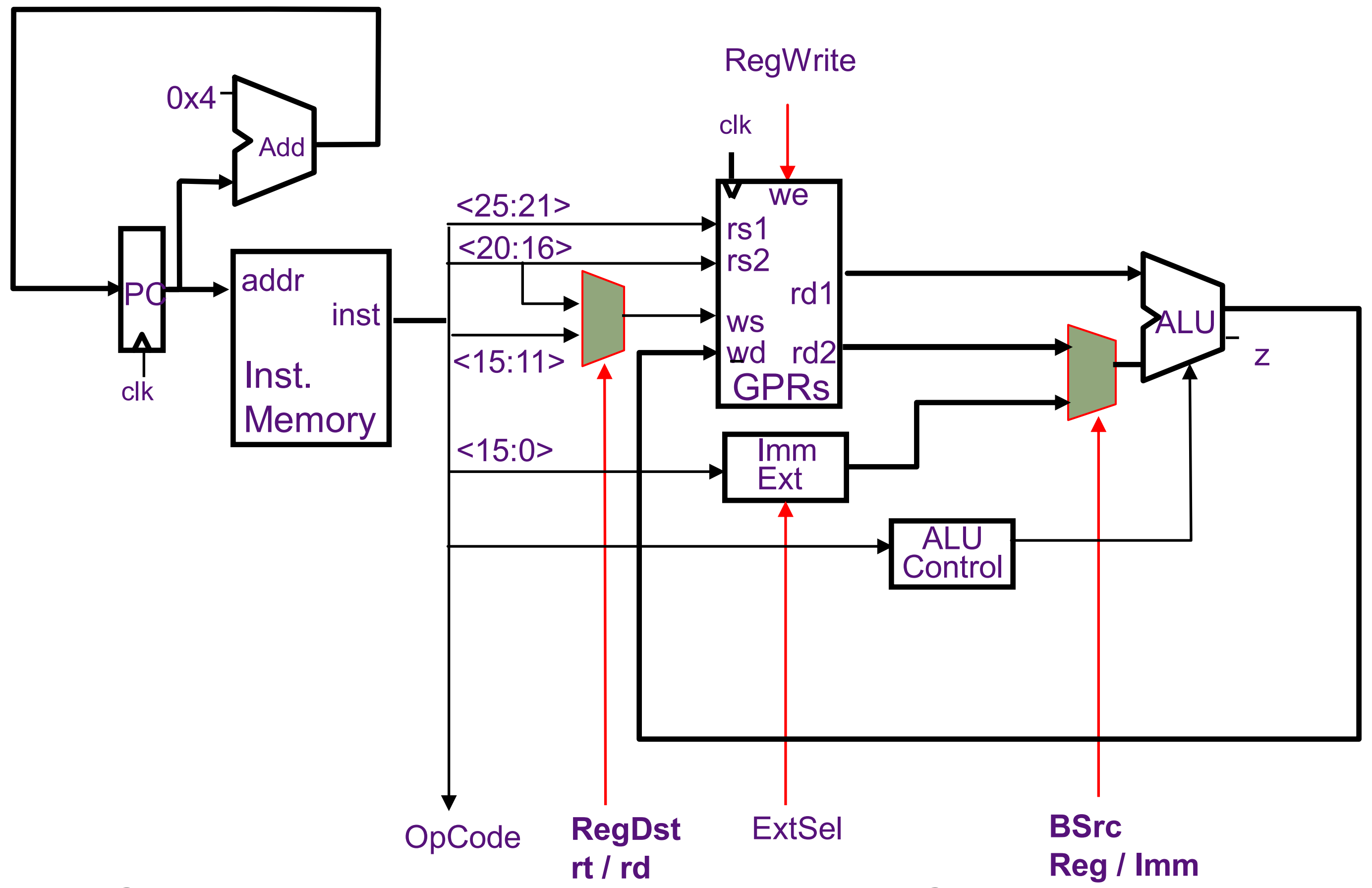
6	5	5	5	5	6	
0	rs	rt	rd	0	func	rd ← (rs) func (rt)
opcode	rs	rt	immediate			rt ← (rs) op immediate

Datapath for 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

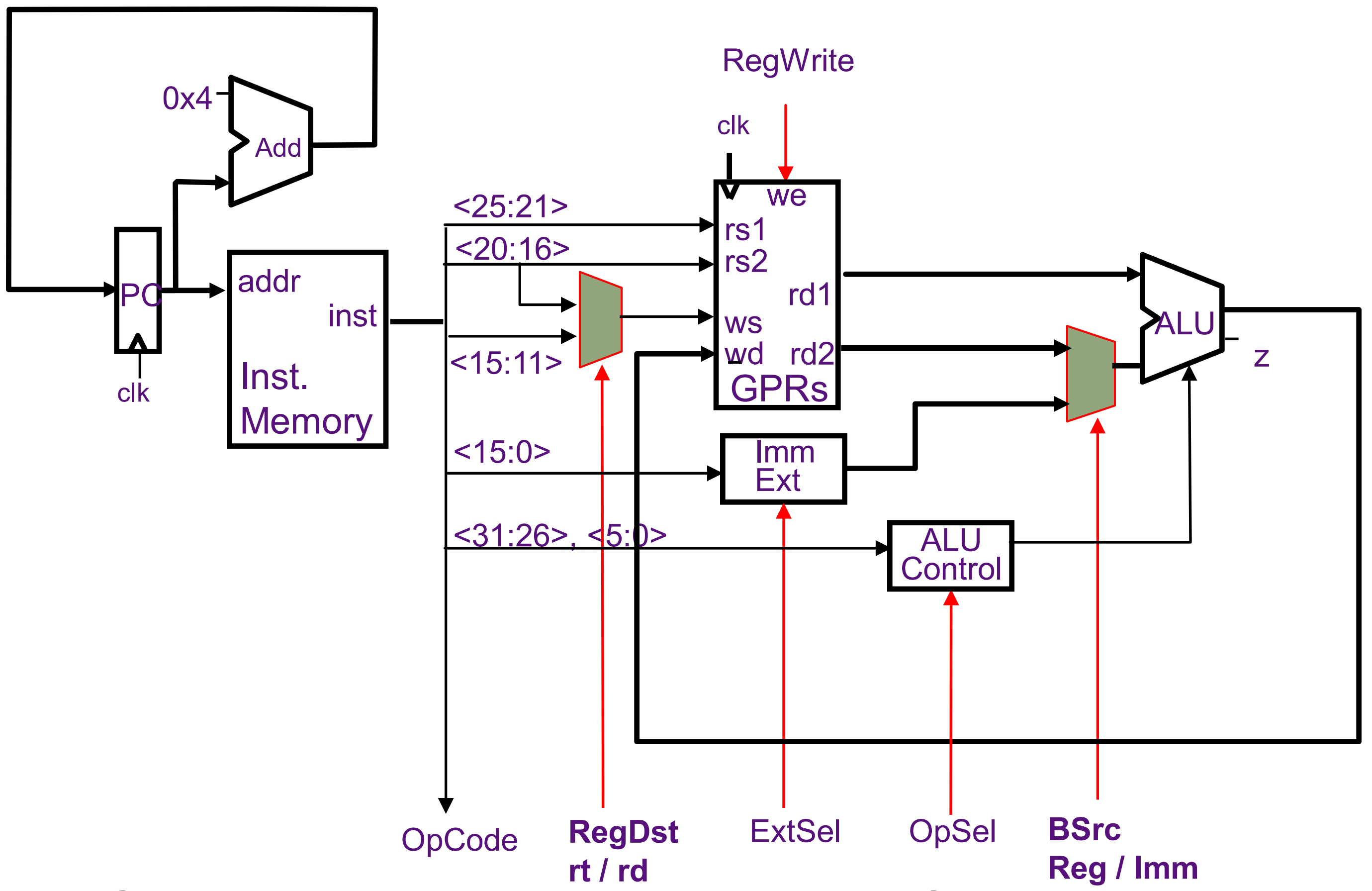
Datapath for ALU Instructions



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 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

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

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

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

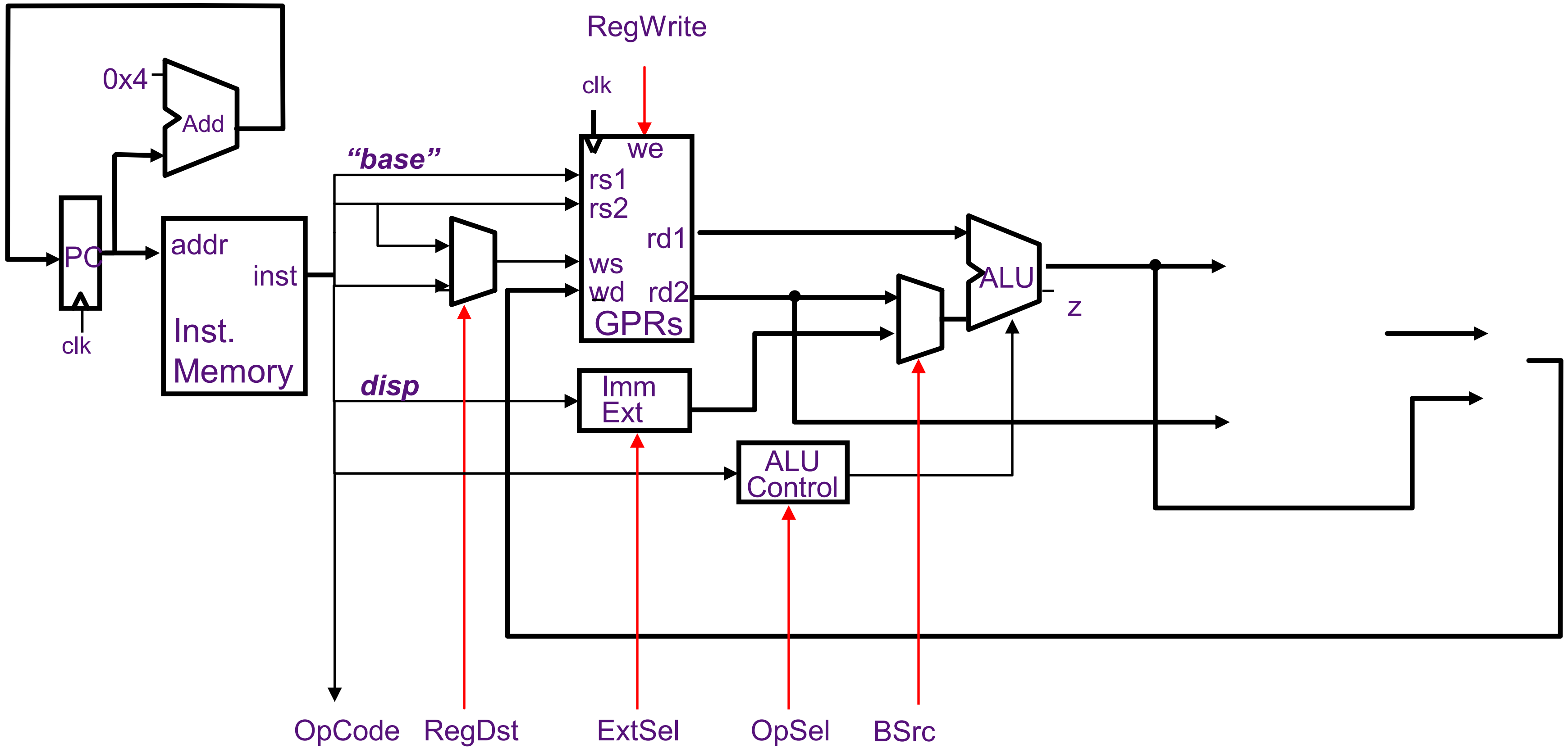
6	5	5	16	addressing mode
opcode	rs	rt	displacement	(rs) + displacement
31 26	25 21	20 16	15 0	

rs is the base register

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

Load/Store Instructions

Harvard Datapath



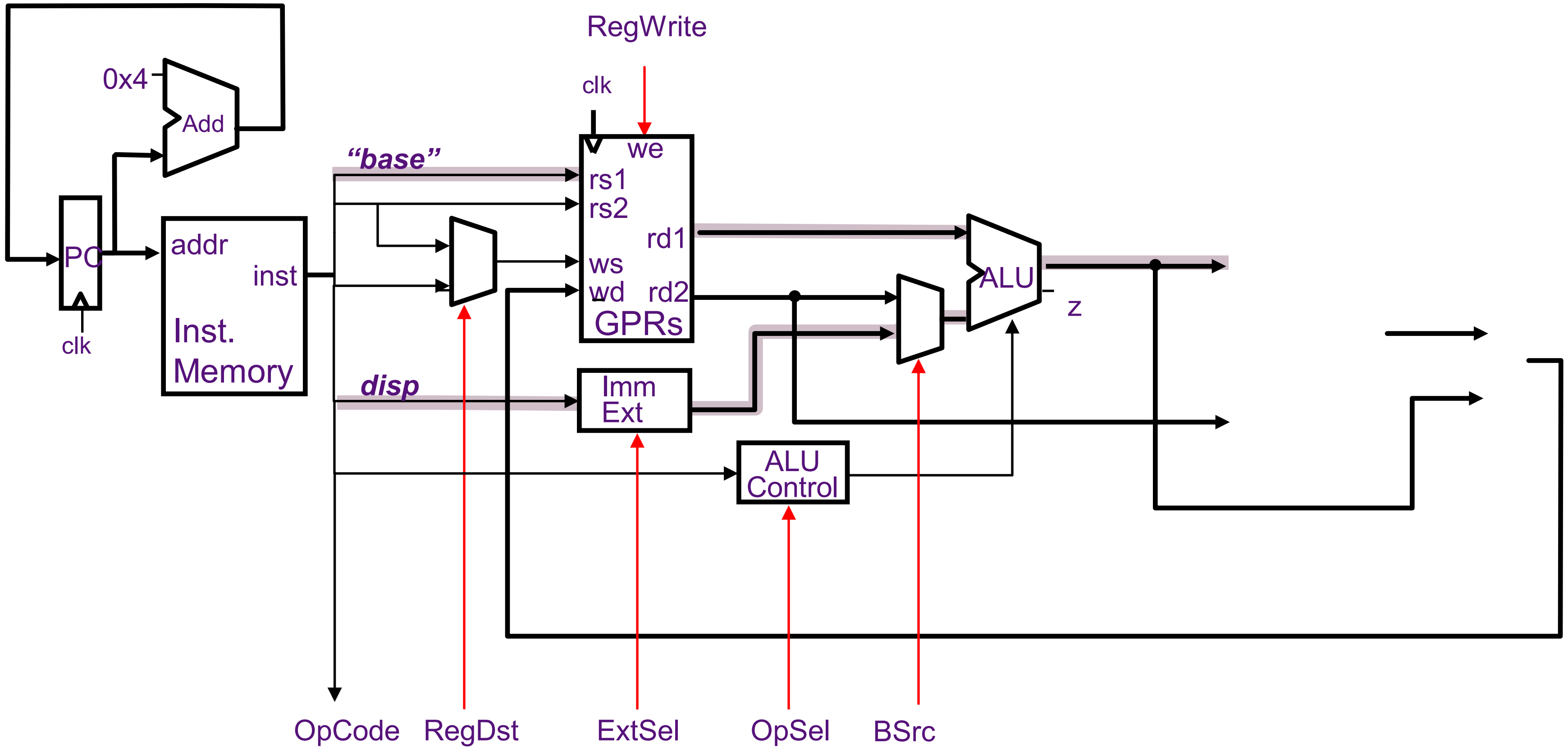
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

Load/Store Instructions

Harvard Datapath



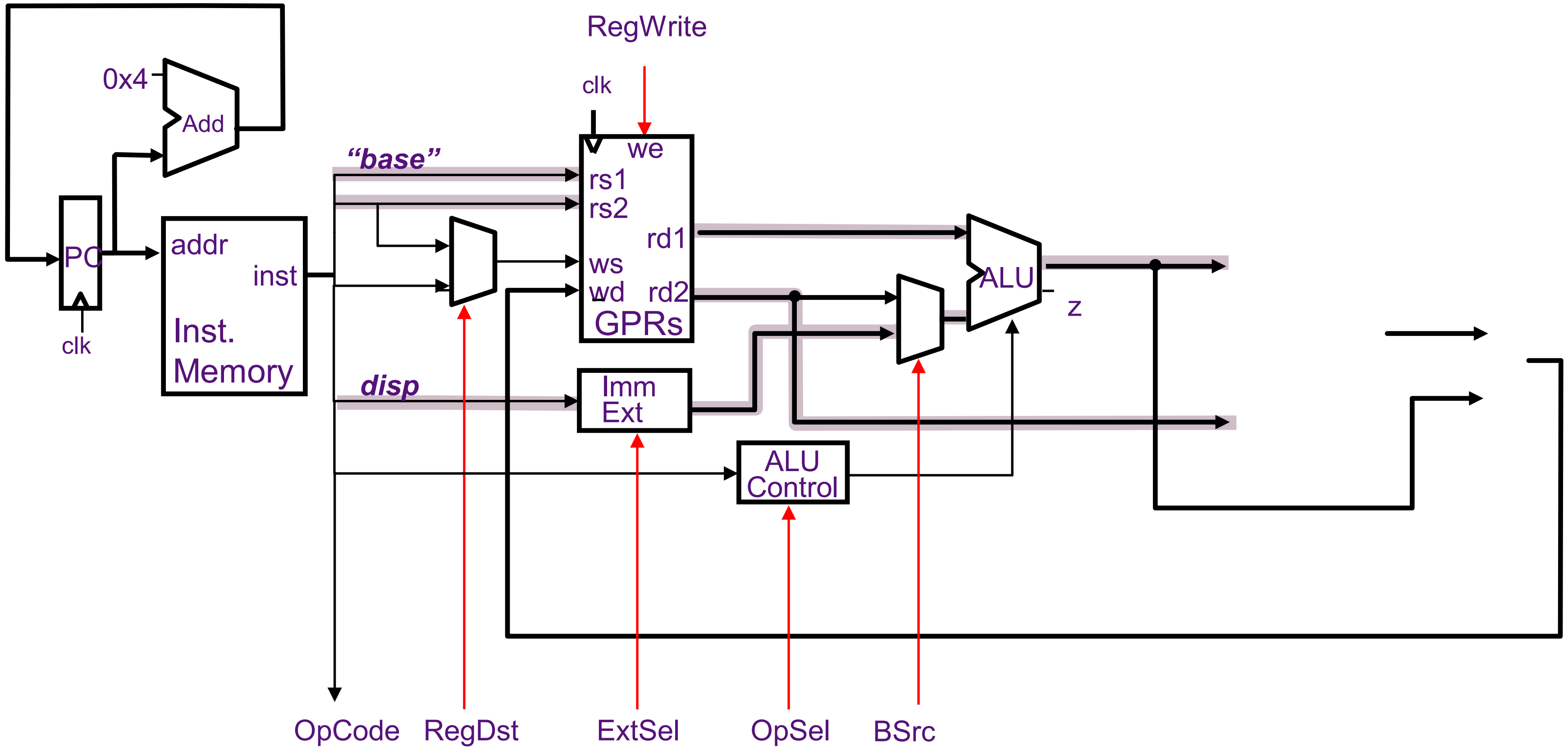
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

Load/Store Instructions

Harvard Datapath



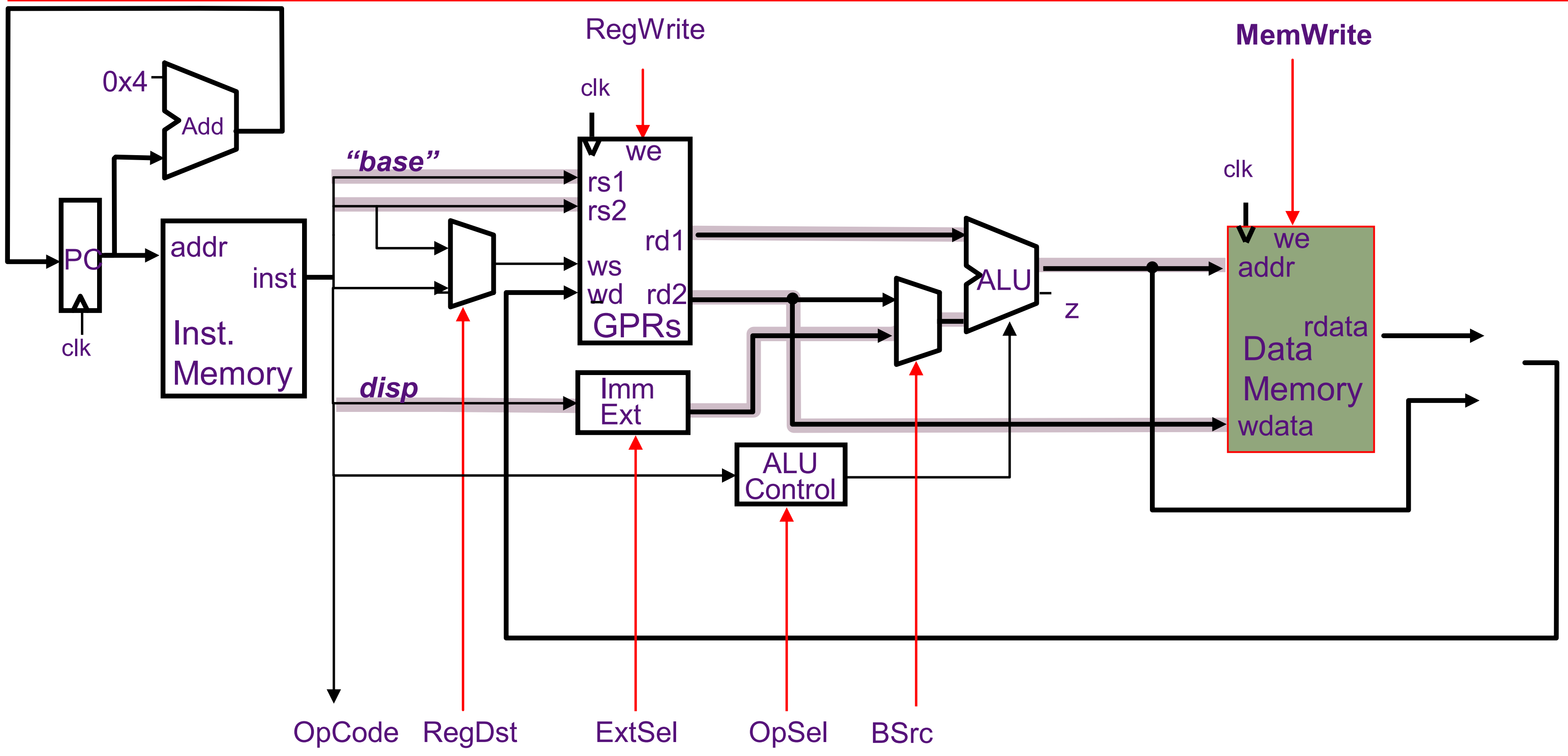
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

Load/Store Instructions

Harvard Datapath



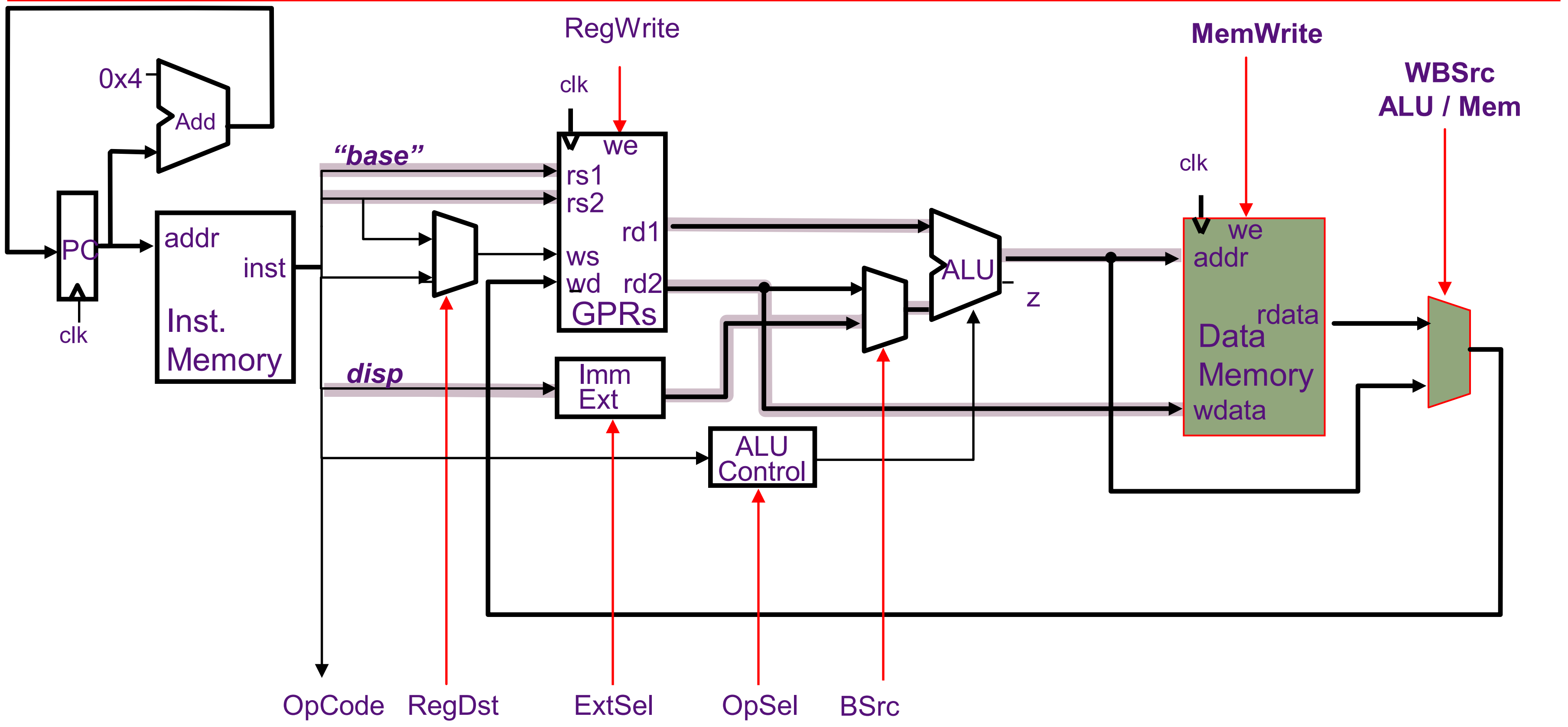
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

Load/Store Instructions

Harvard Datapath



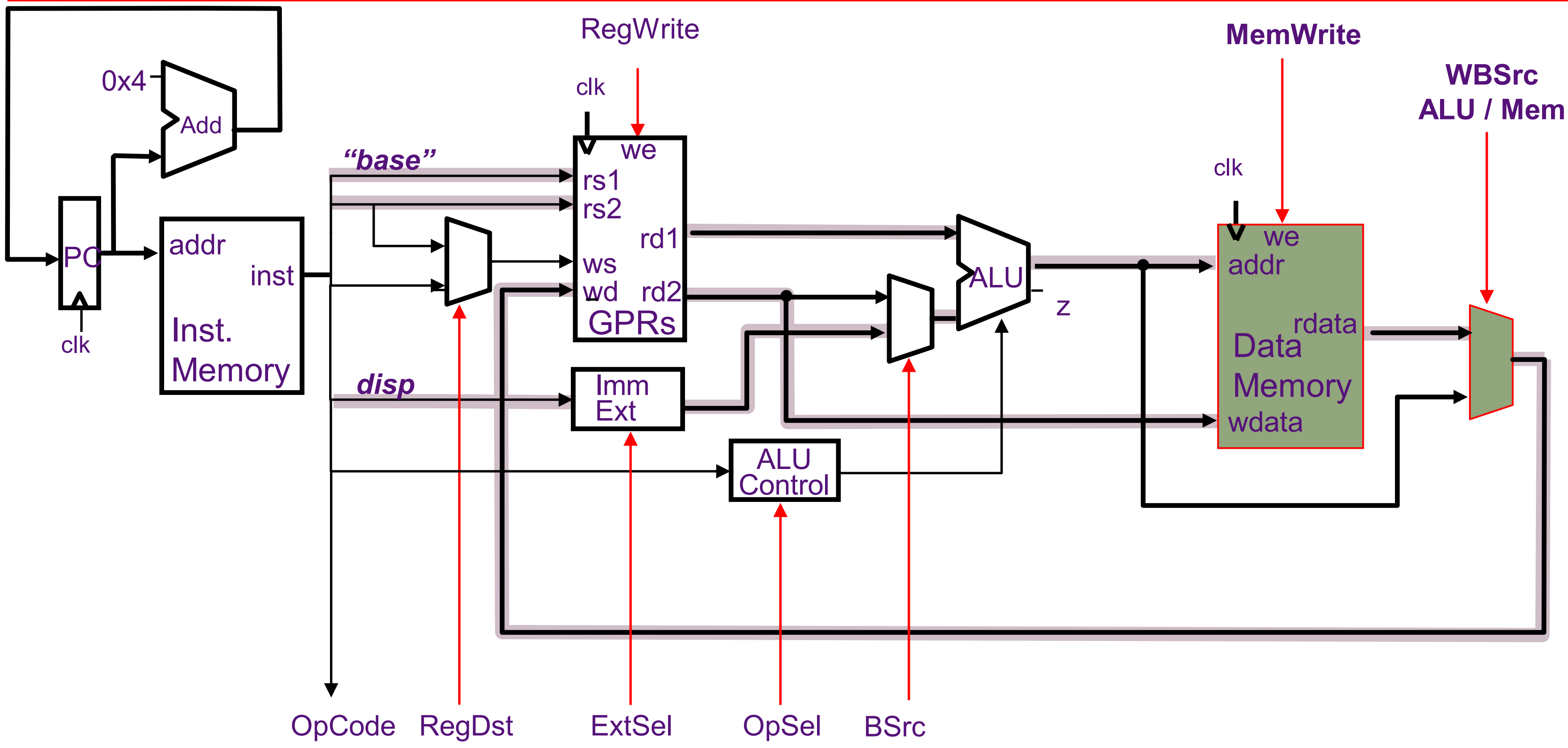
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

Load/Store Instructions

Harvard Datapath



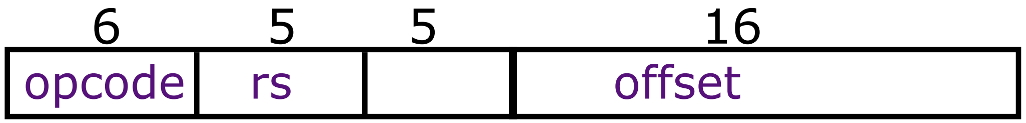
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

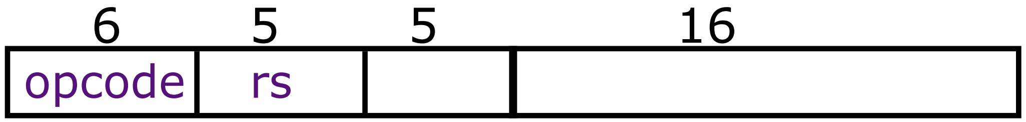
MIPS Control Instructions

Conditional (on GPR) PC-relative branch



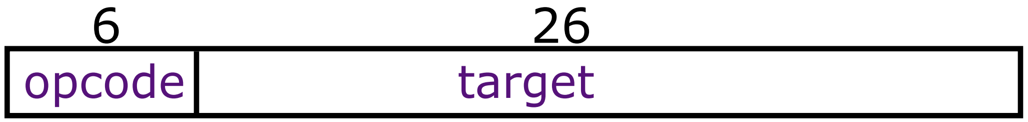
BEQZ, BNEZ

Unconditional register-indirect jumps



JR, JALR

Unconditional absolute jumps

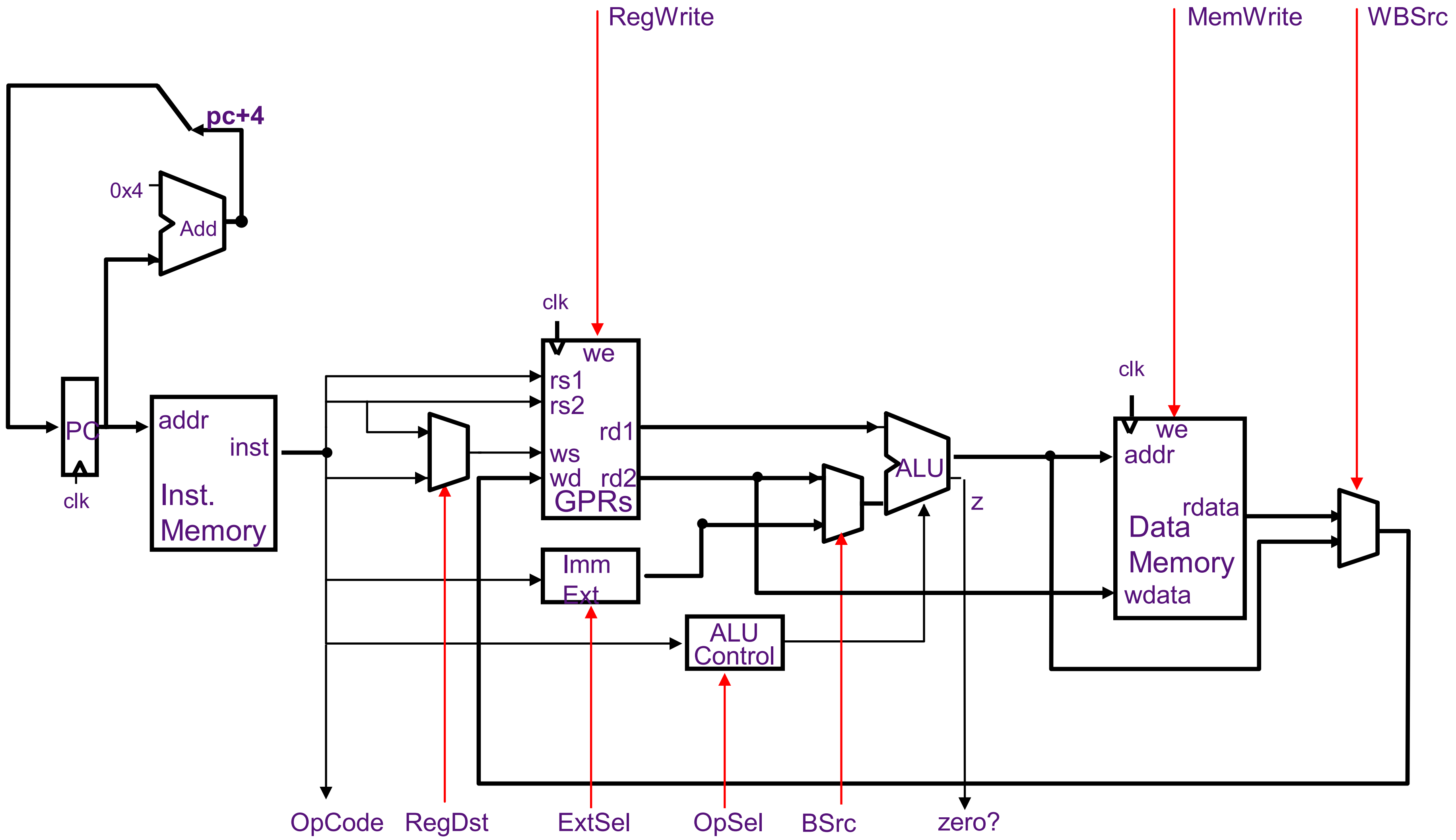


J, JAL

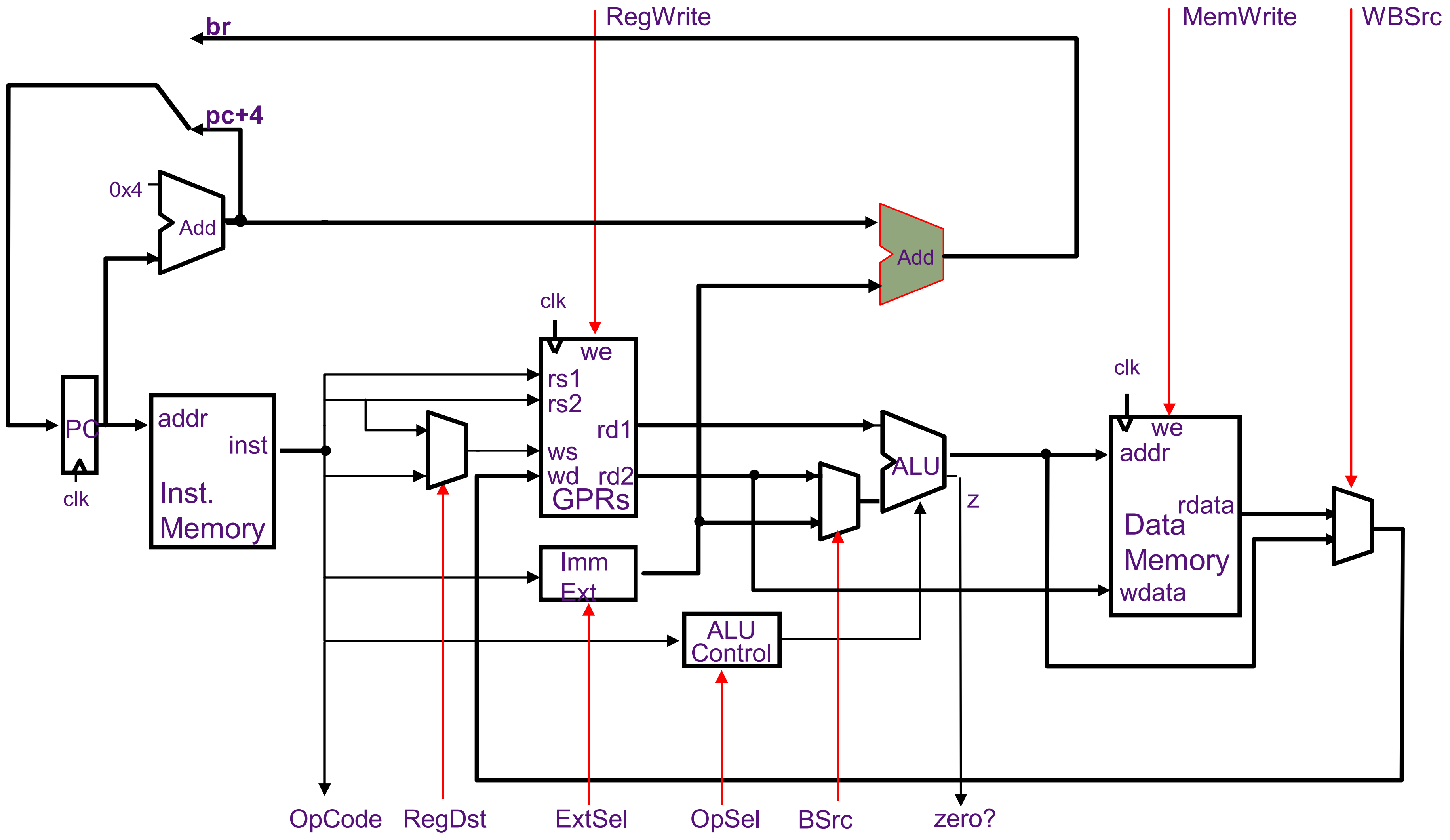
- PC-relative branches add offset×4 to PC+4 to calculate the target address (offset is in words): ±128 KB range
- Absolute jumps append target×4 to PC<31:28> to calculate the target address: 256 MB range
- Jump-&-link stores PC+4 into the link register (R31)
- Control transfers are not delayed

  we will worry about the branch delay slot later

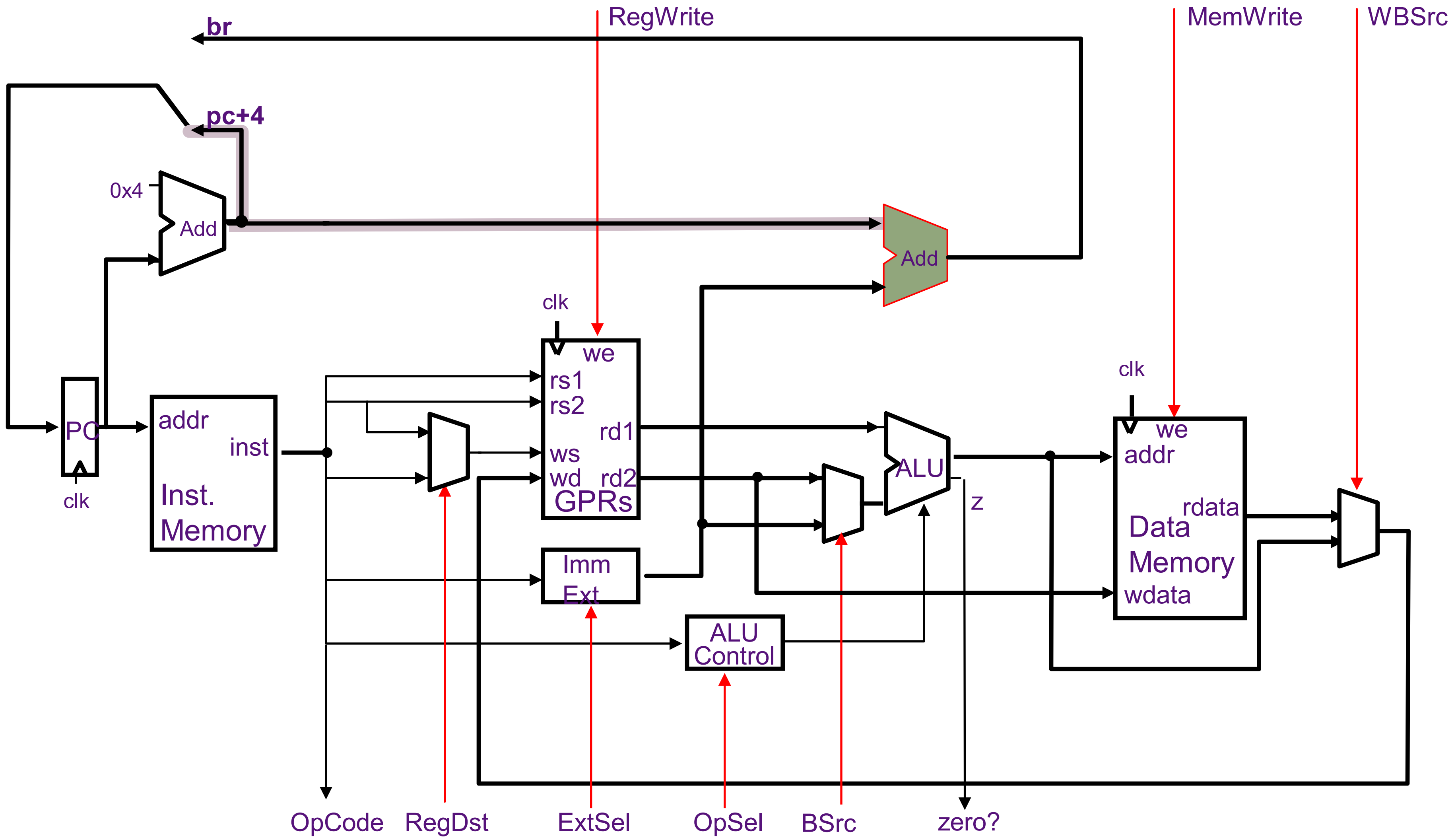
Conditional Branches (BEQZ, BNEZ)



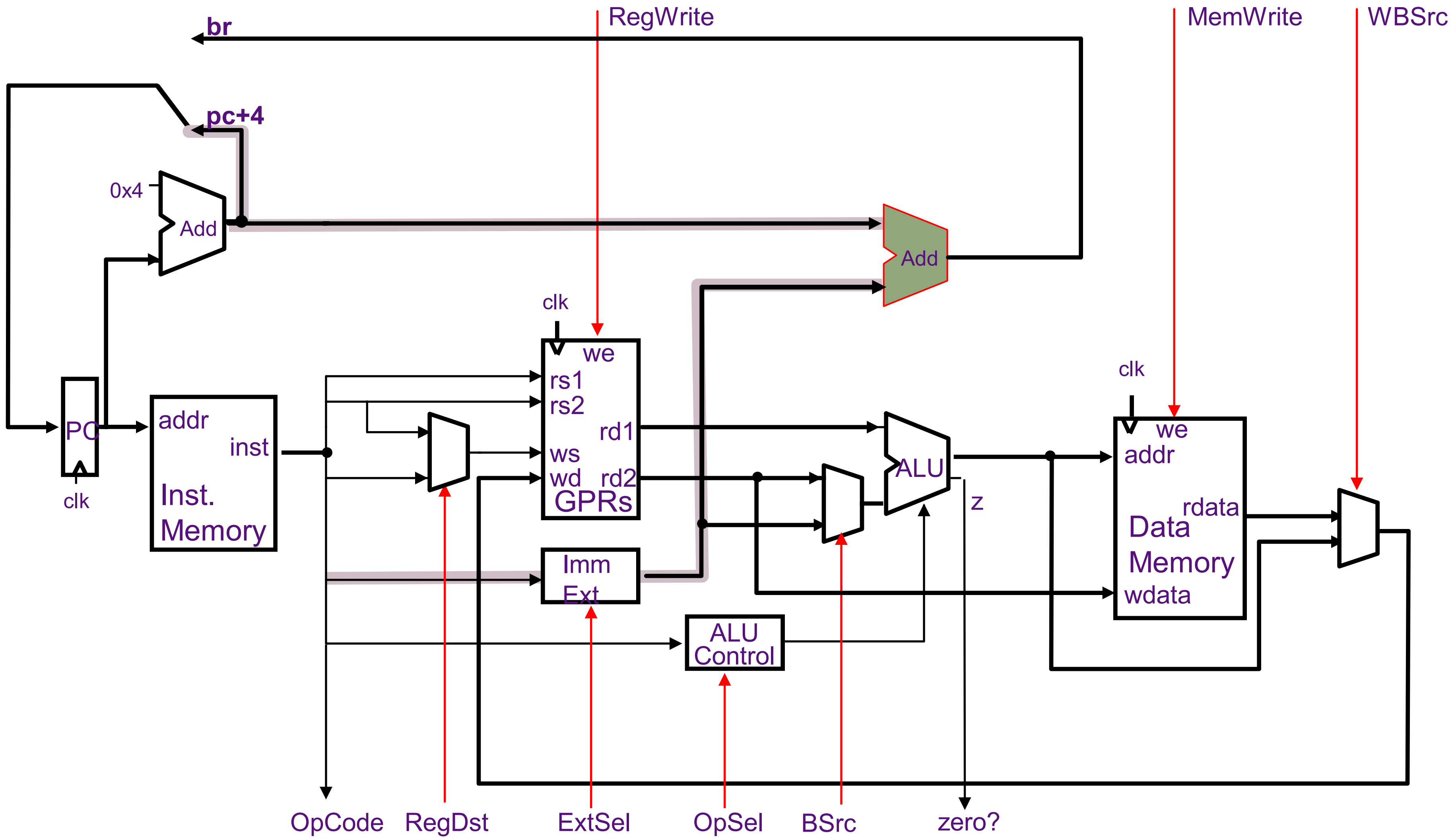
Conditional Branches (BEQZ, BNEZ)



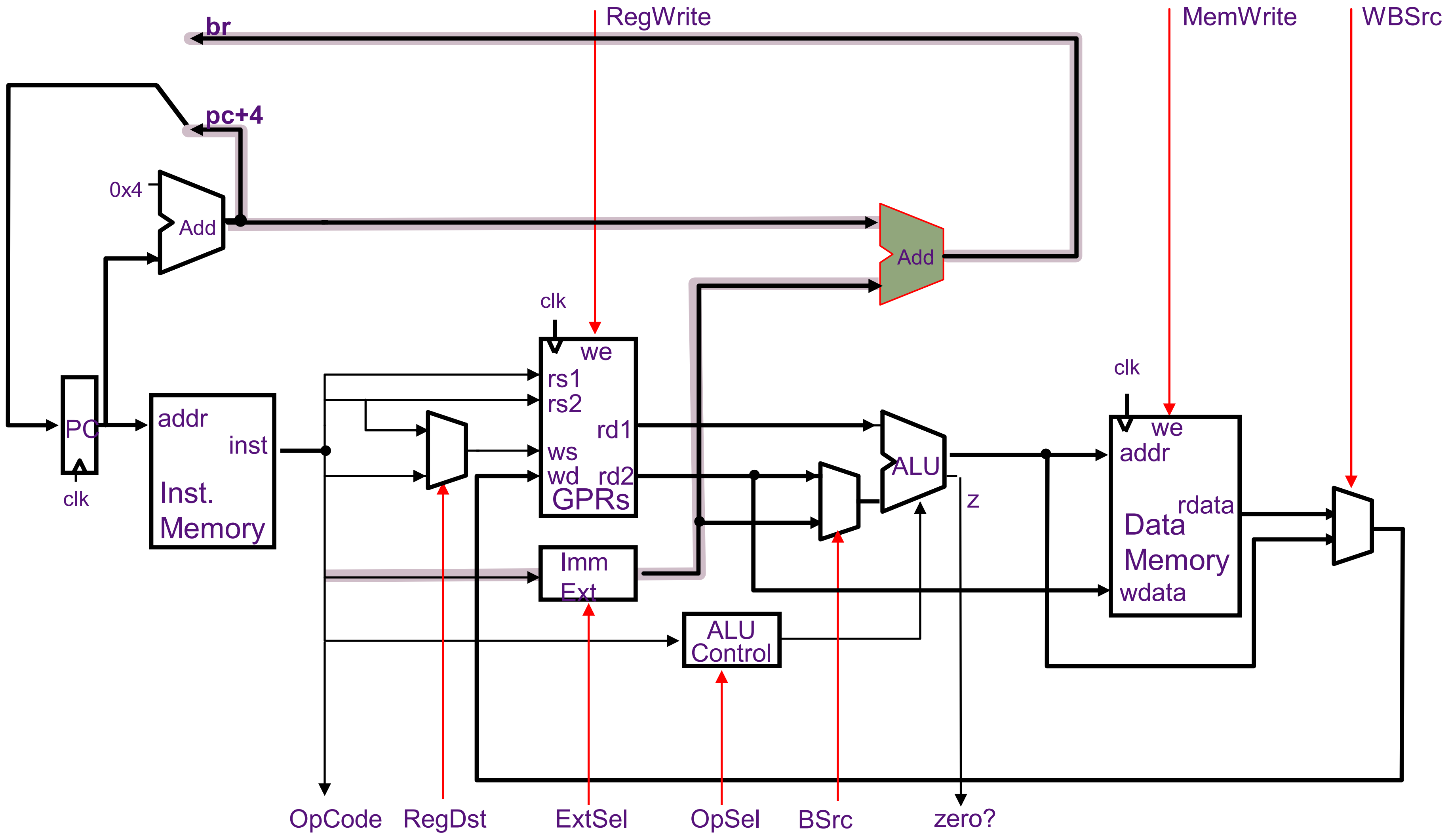
Conditional Branches (BEQZ, BNEZ)



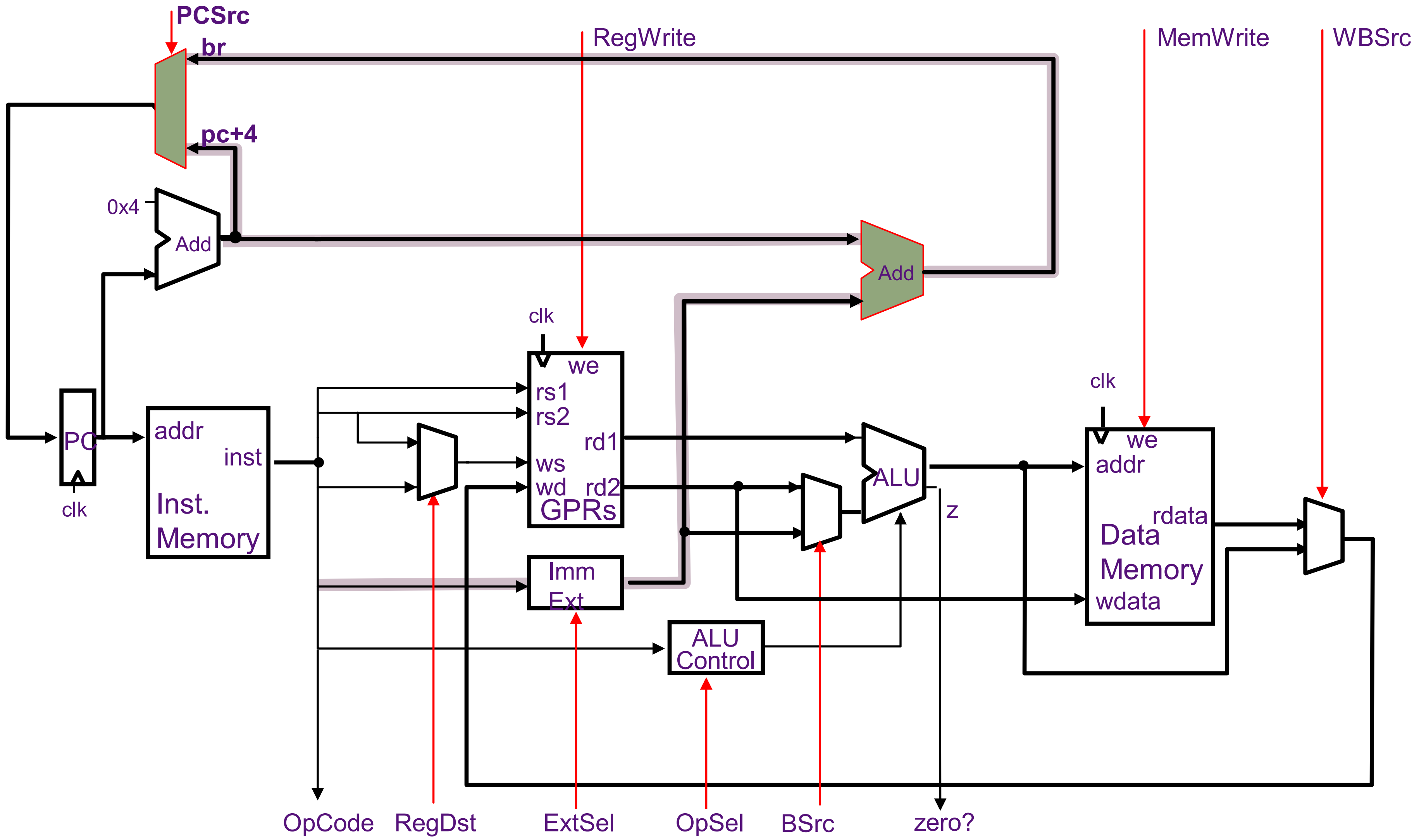
Conditional Branches (BEQZ, BNEZ)



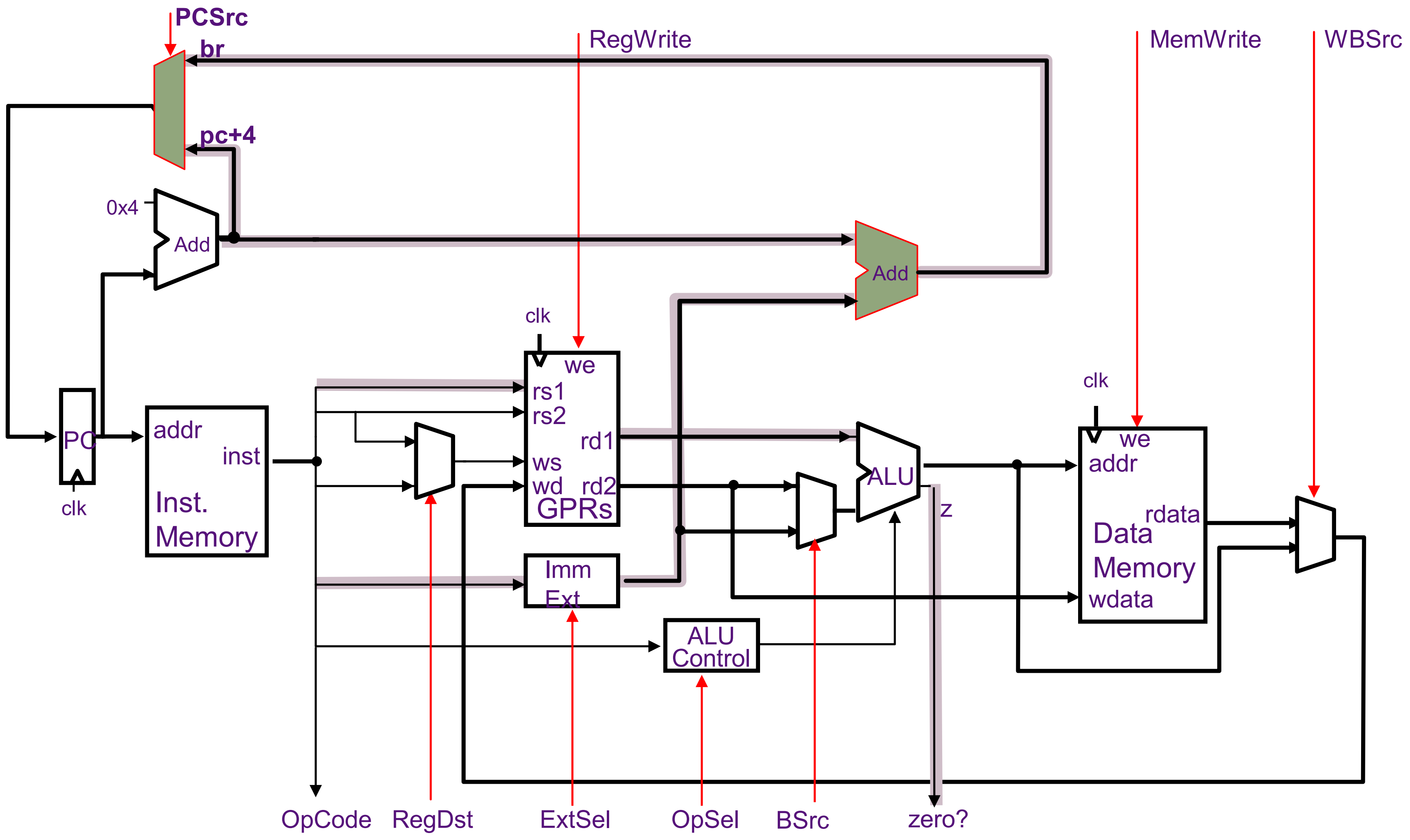
Conditional Branches (BEQZ, BNEZ)



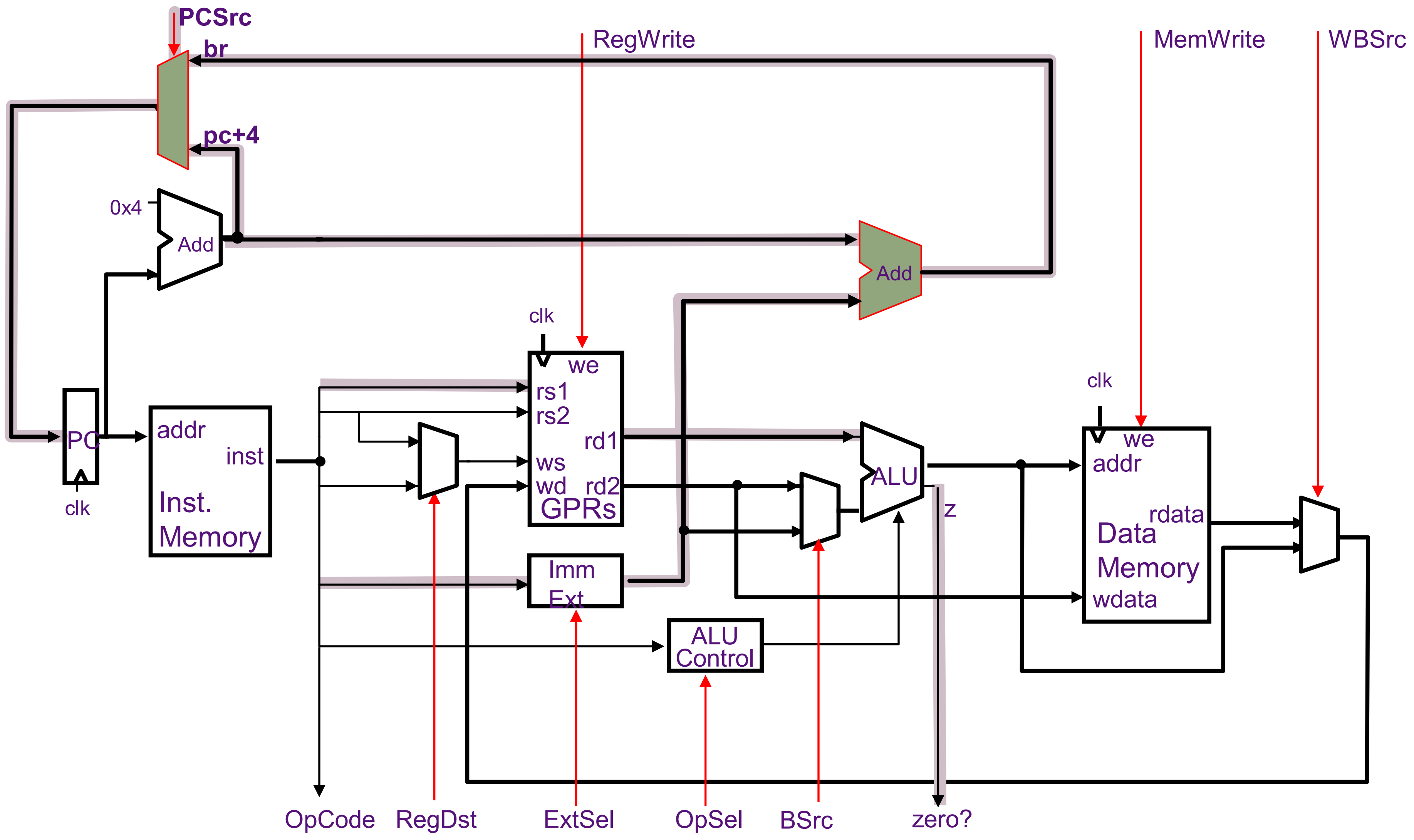
Conditional Branches (BEQZ, BNEZ)



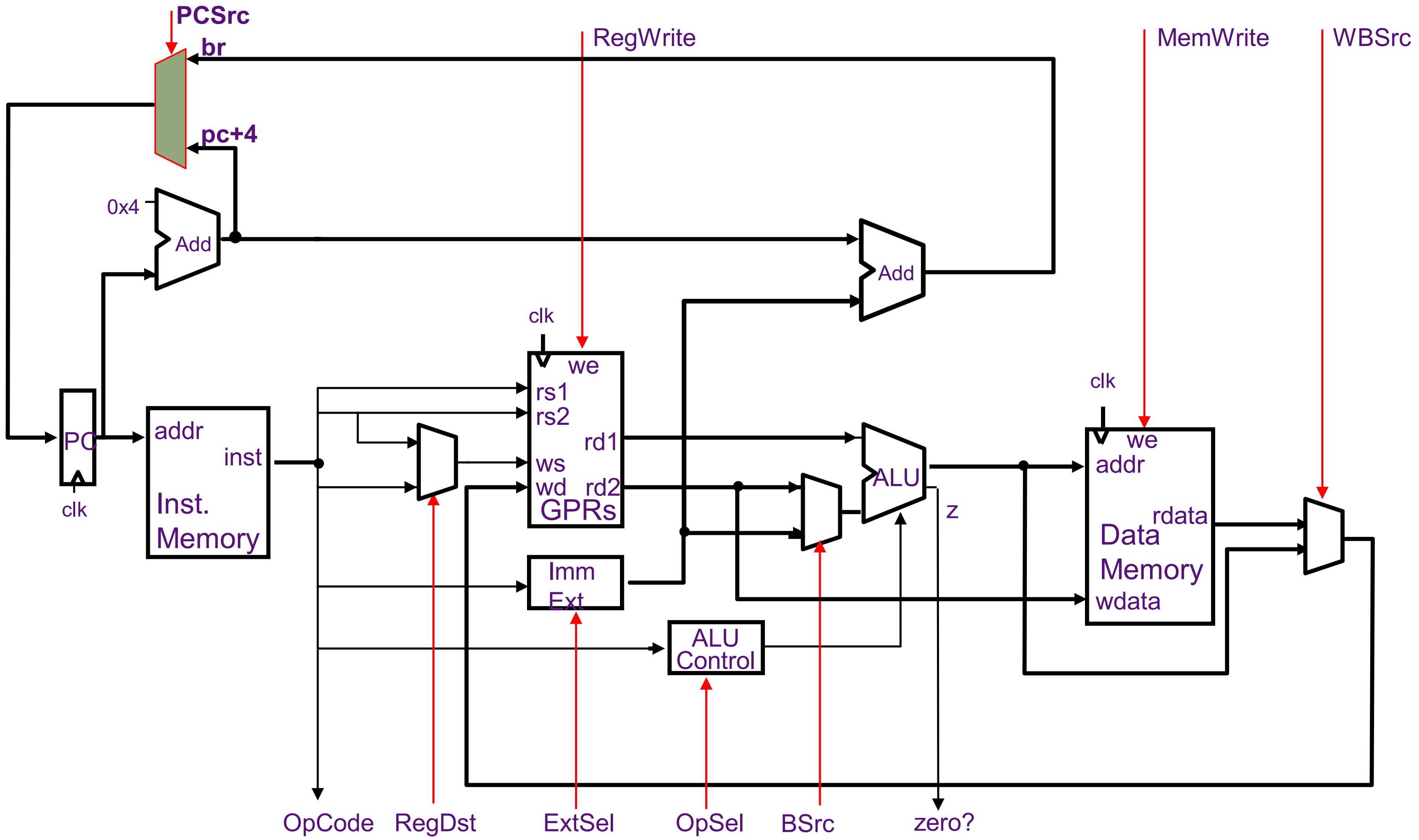
Conditional Branches (BEQZ, BNEZ)



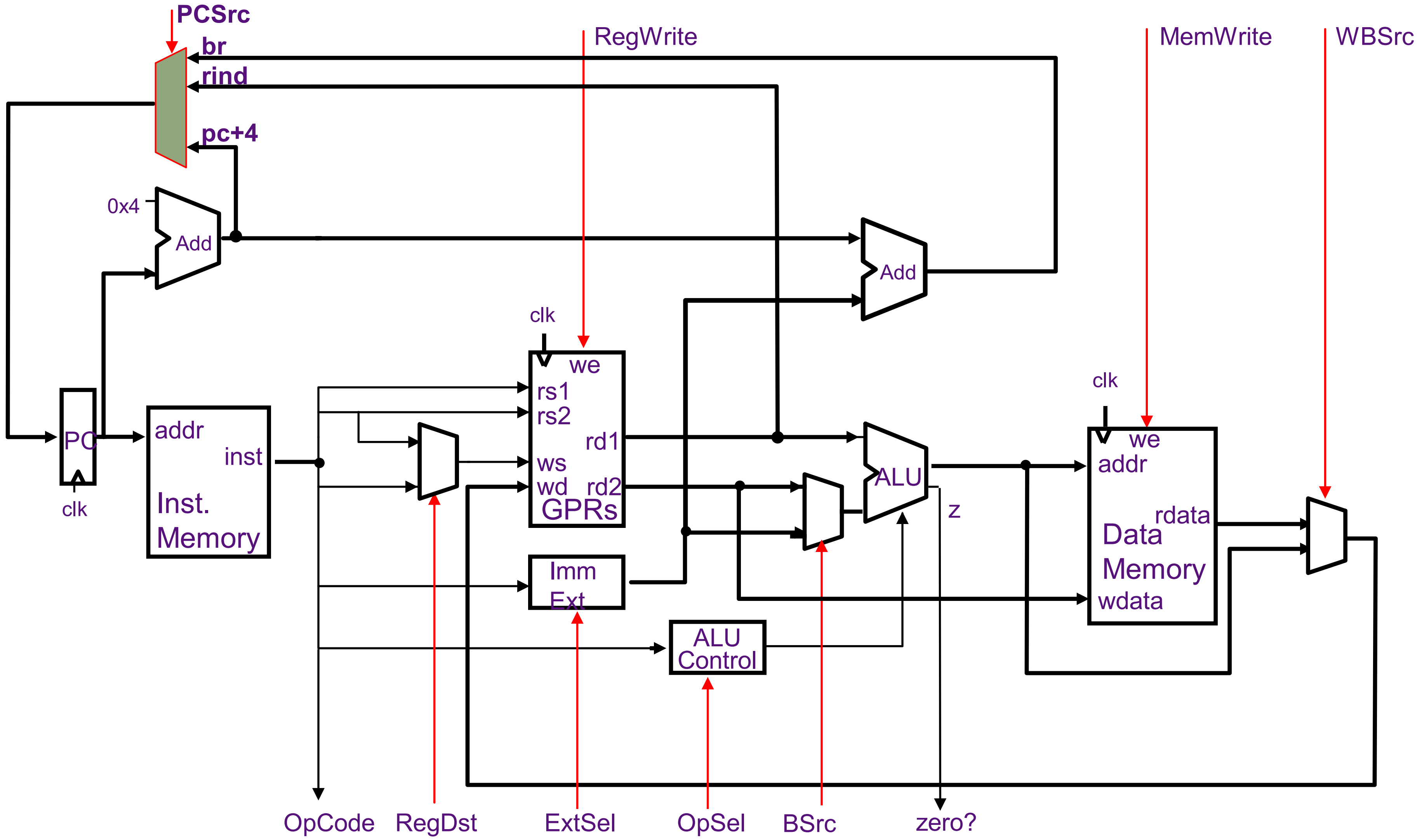
Conditional Branches (BEQZ, BNEZ)



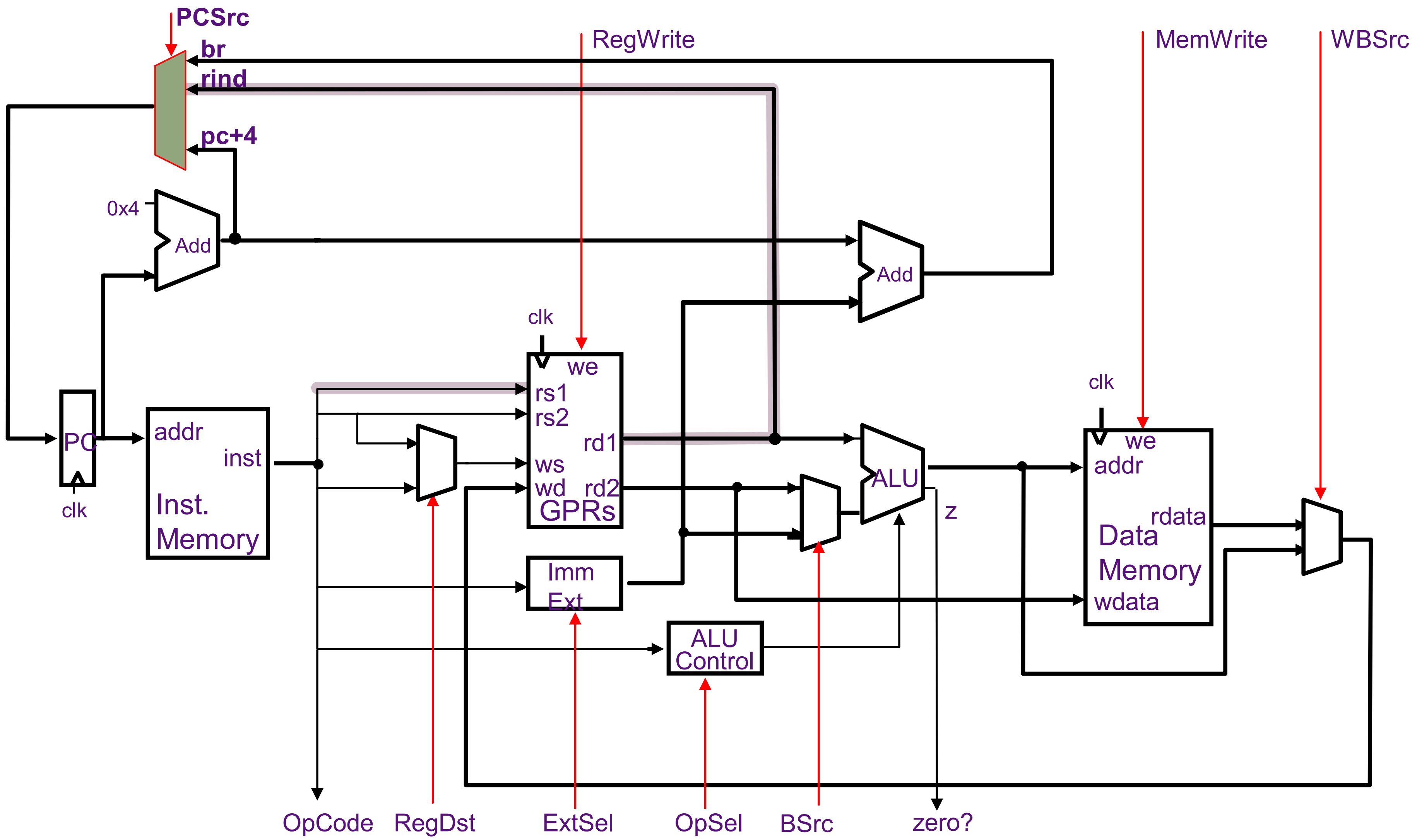
Register-Indirect Jumps (JR)



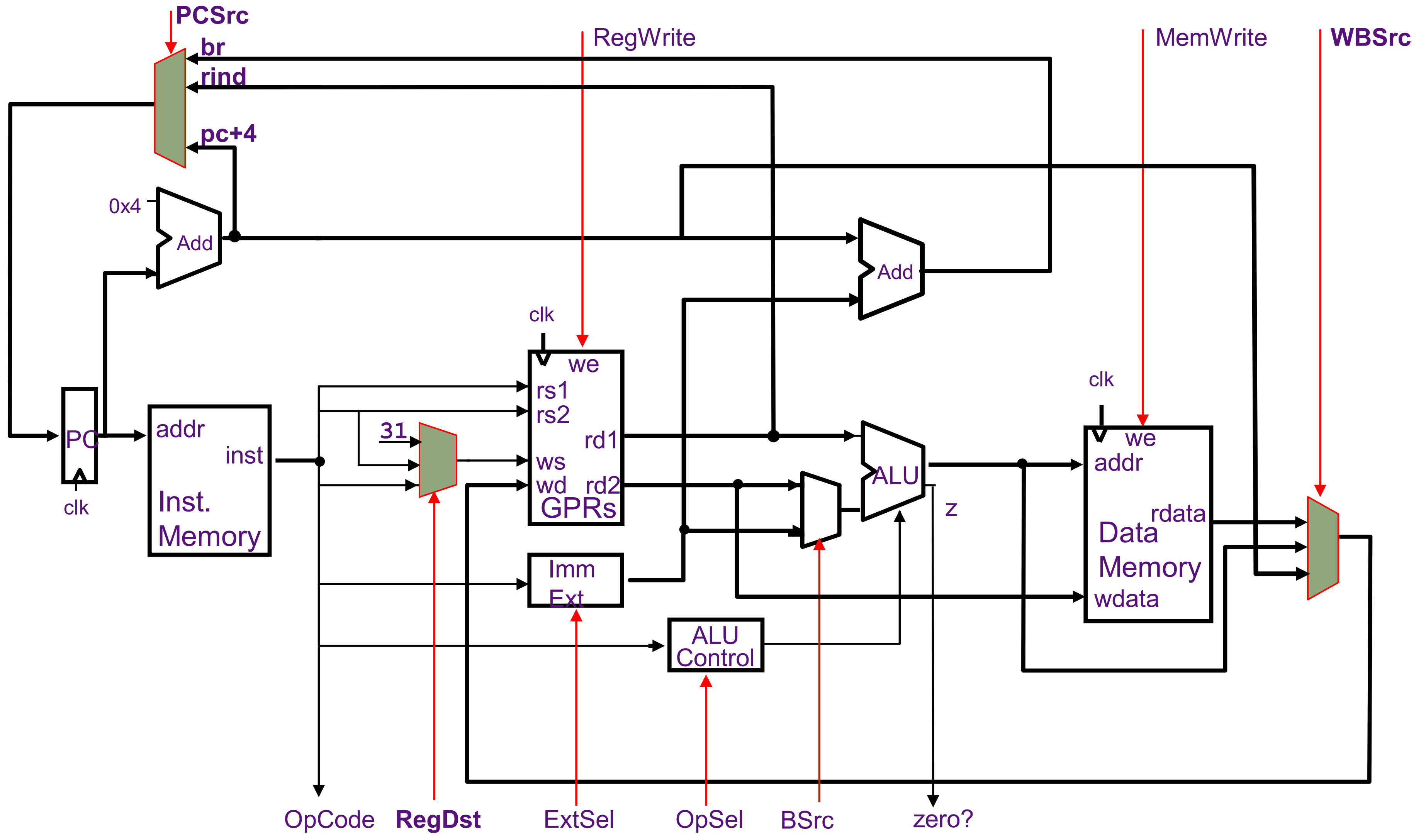
Register-Indirect Jumps (JR)



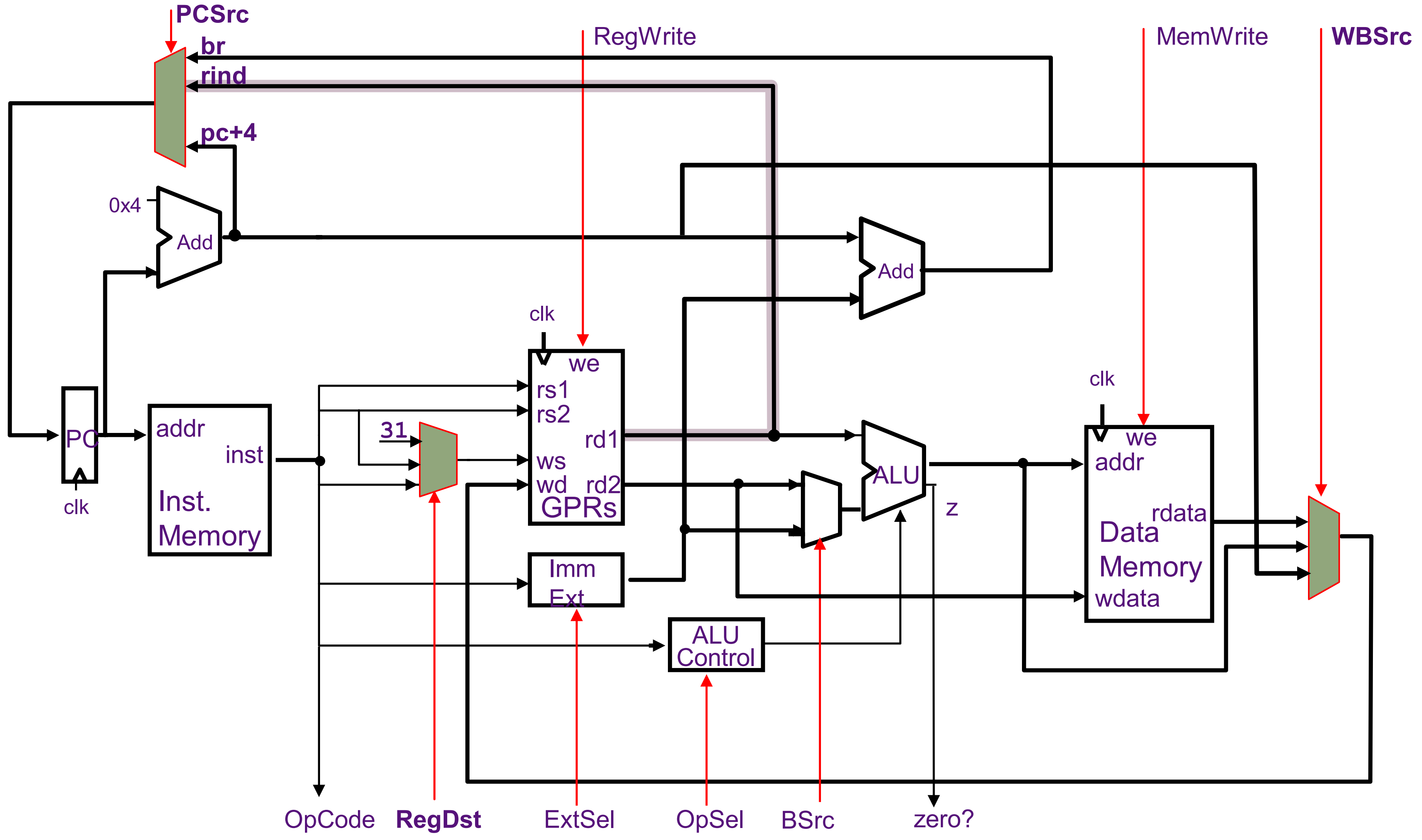
Register-Indirect Jumps (JR)



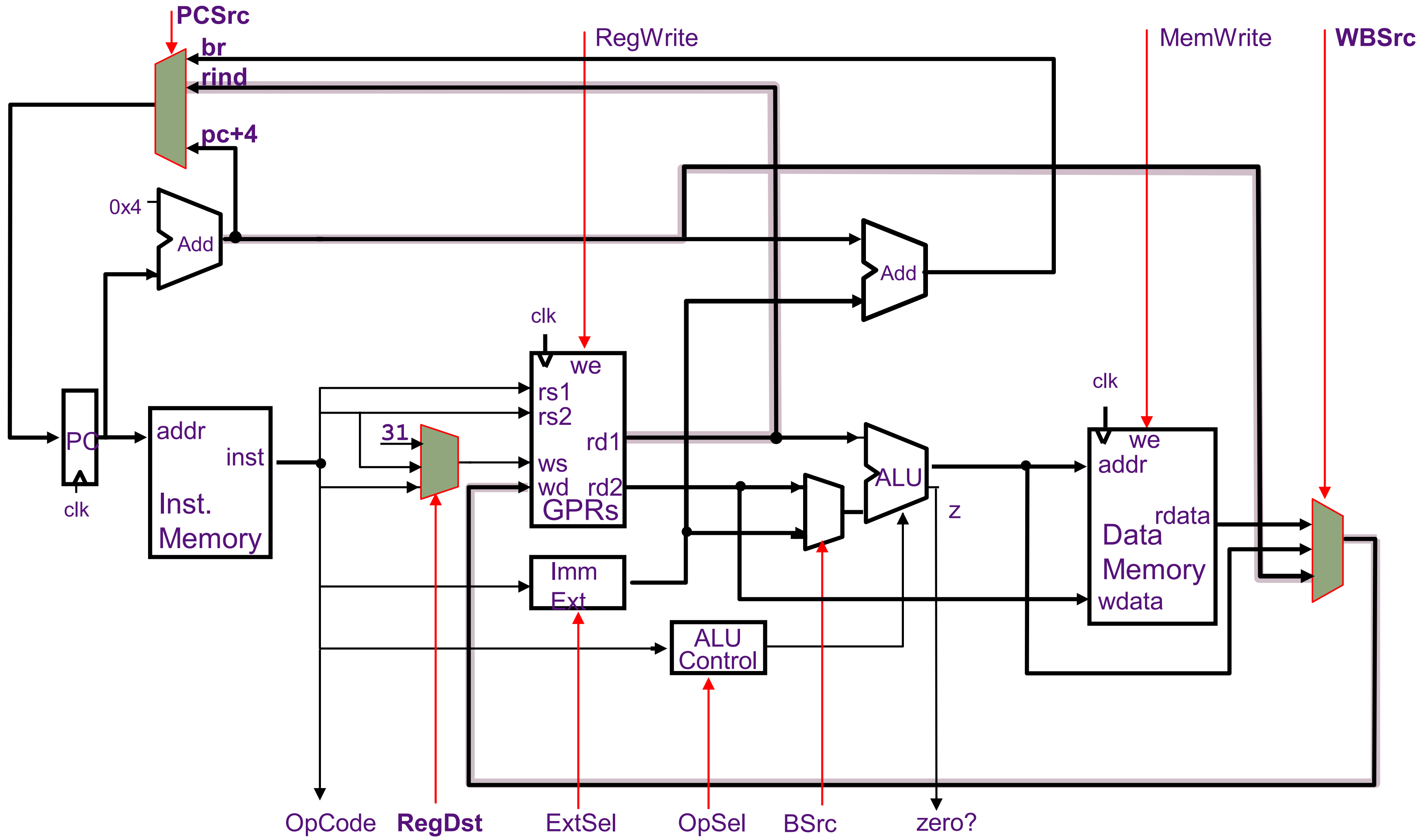
Register-Indirect Jump-&-Link (JALR)



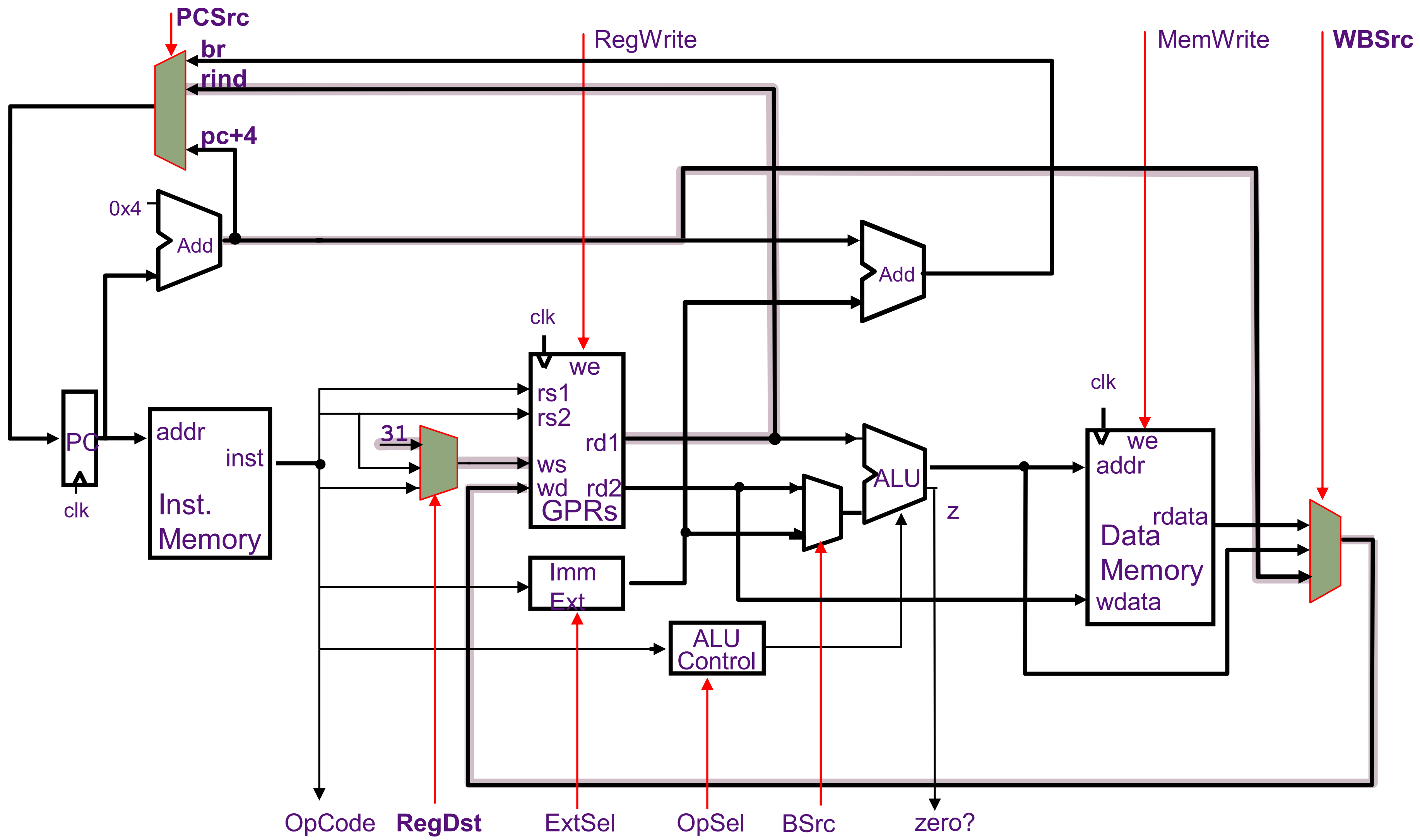
Register-Indirect Jump-&-Link (JALR)



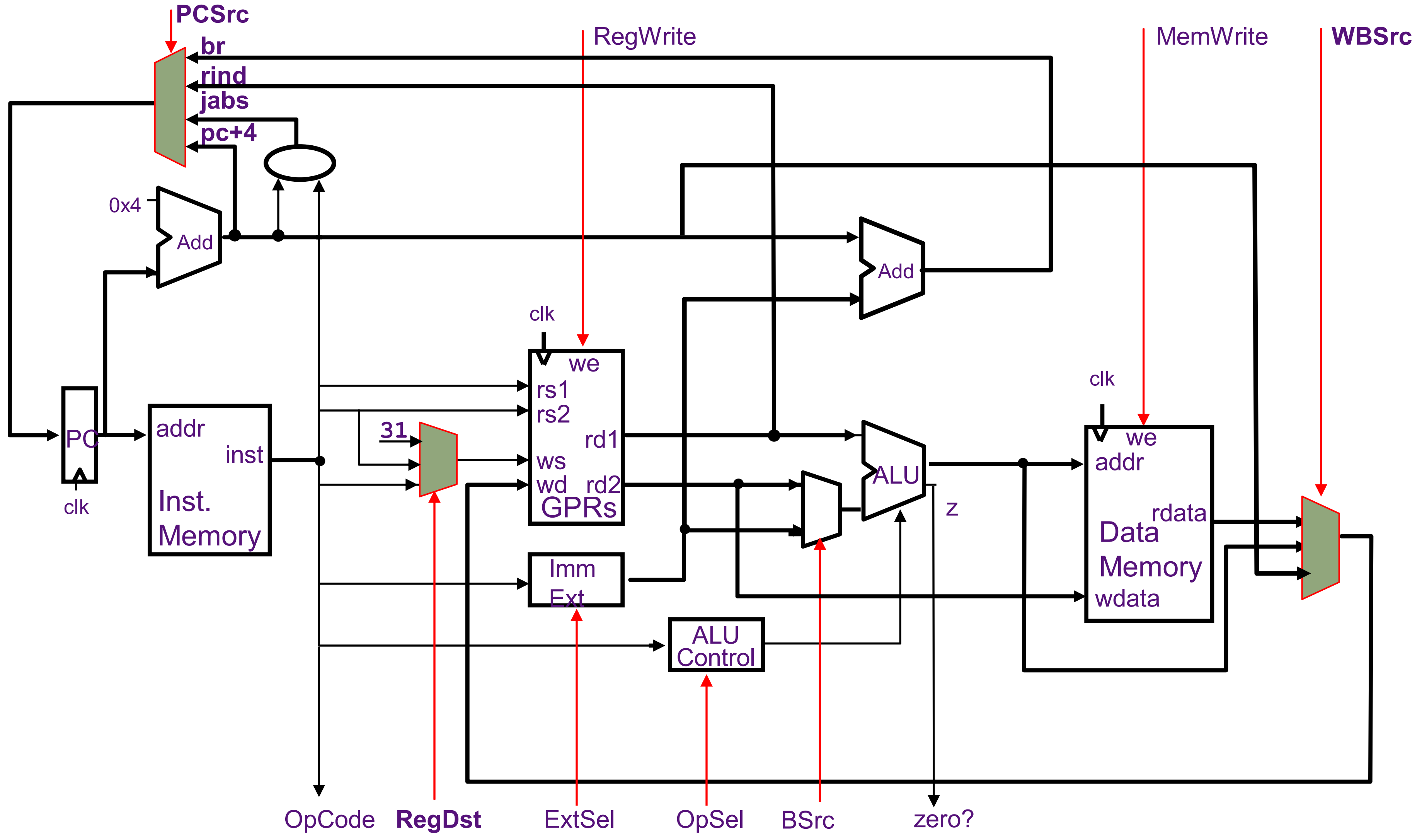
Register-Indirect Jump-&-Link (JALR)



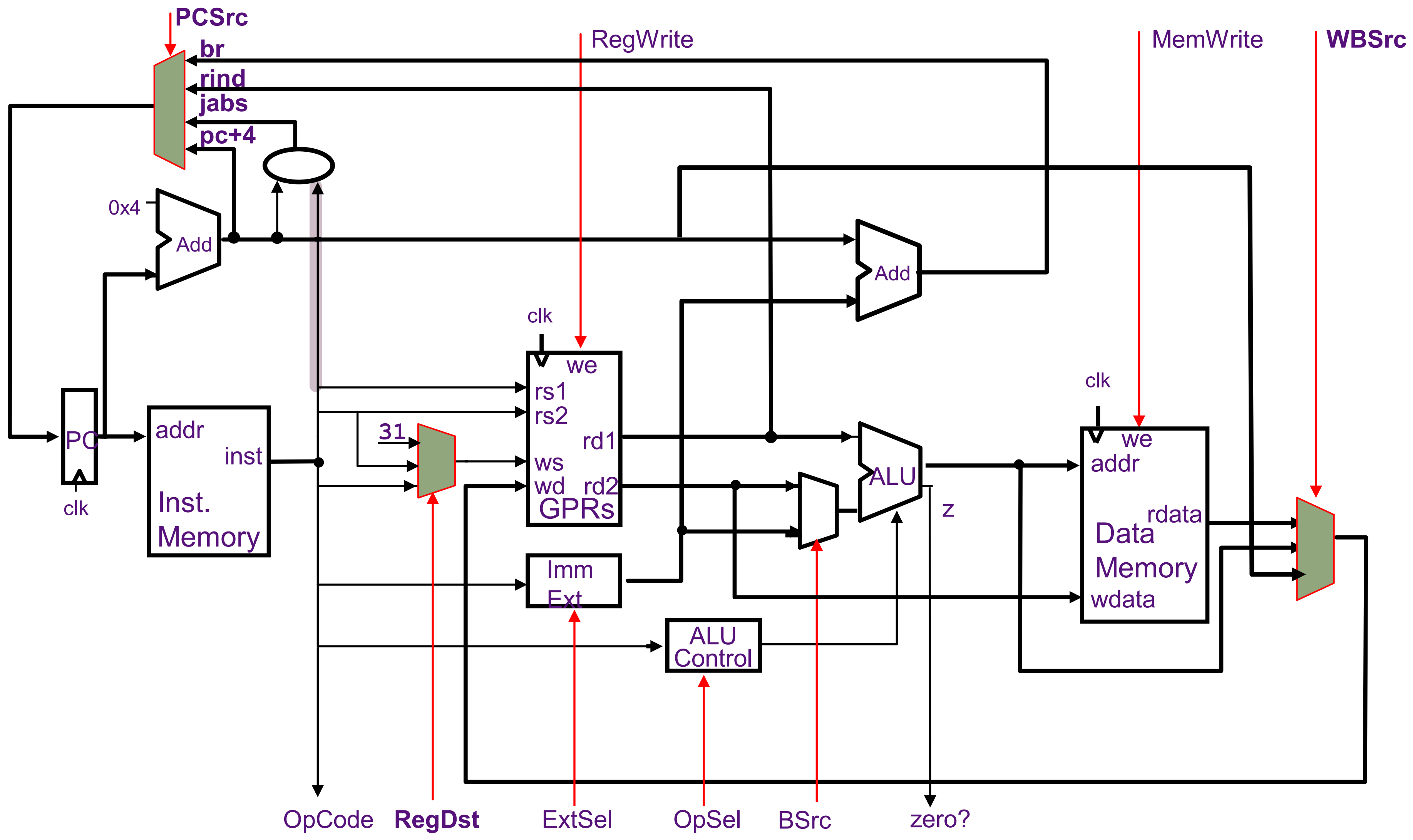
Register-Indirect Jump-&-Link (JALR)



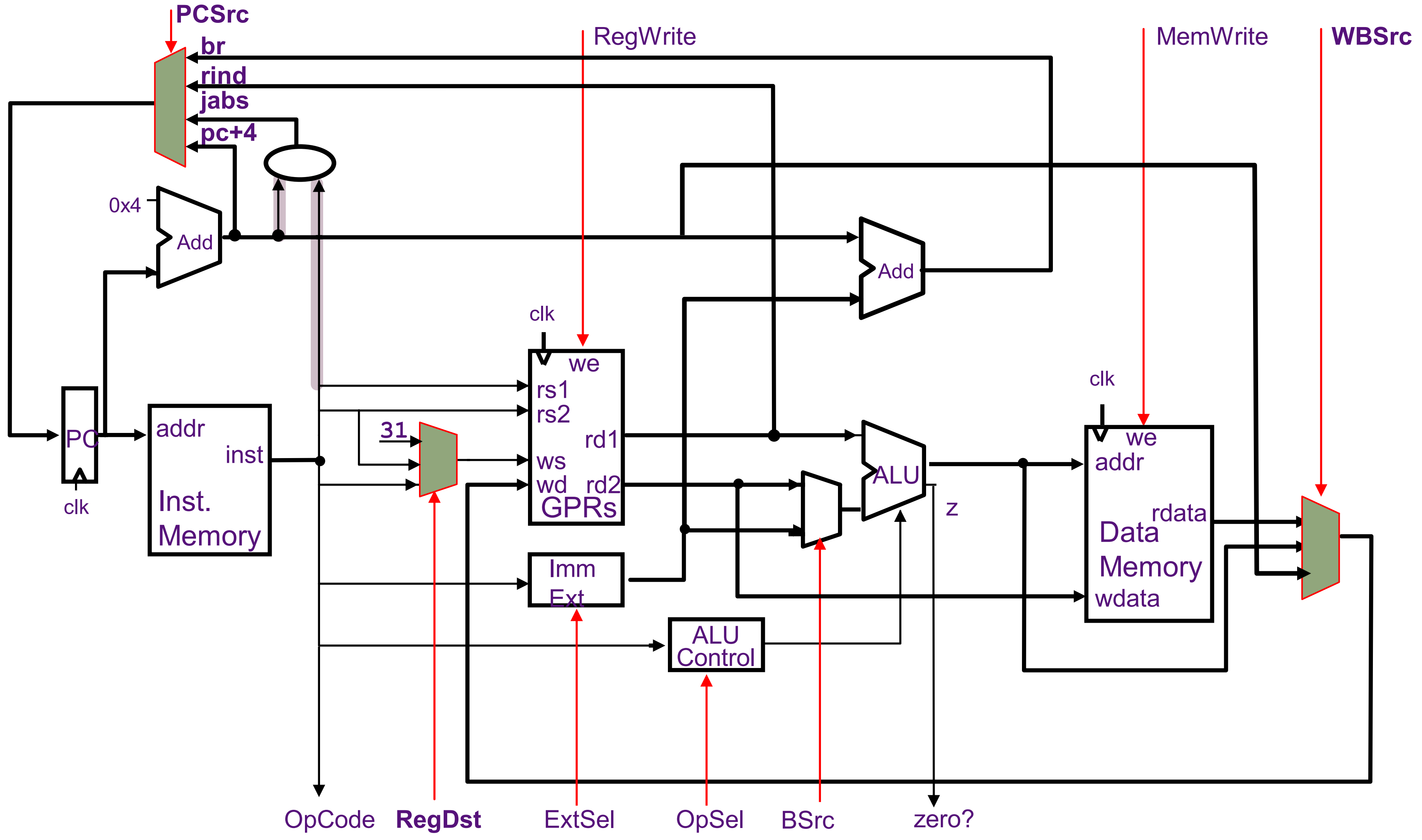
Absolute Jumps (J, JAL)



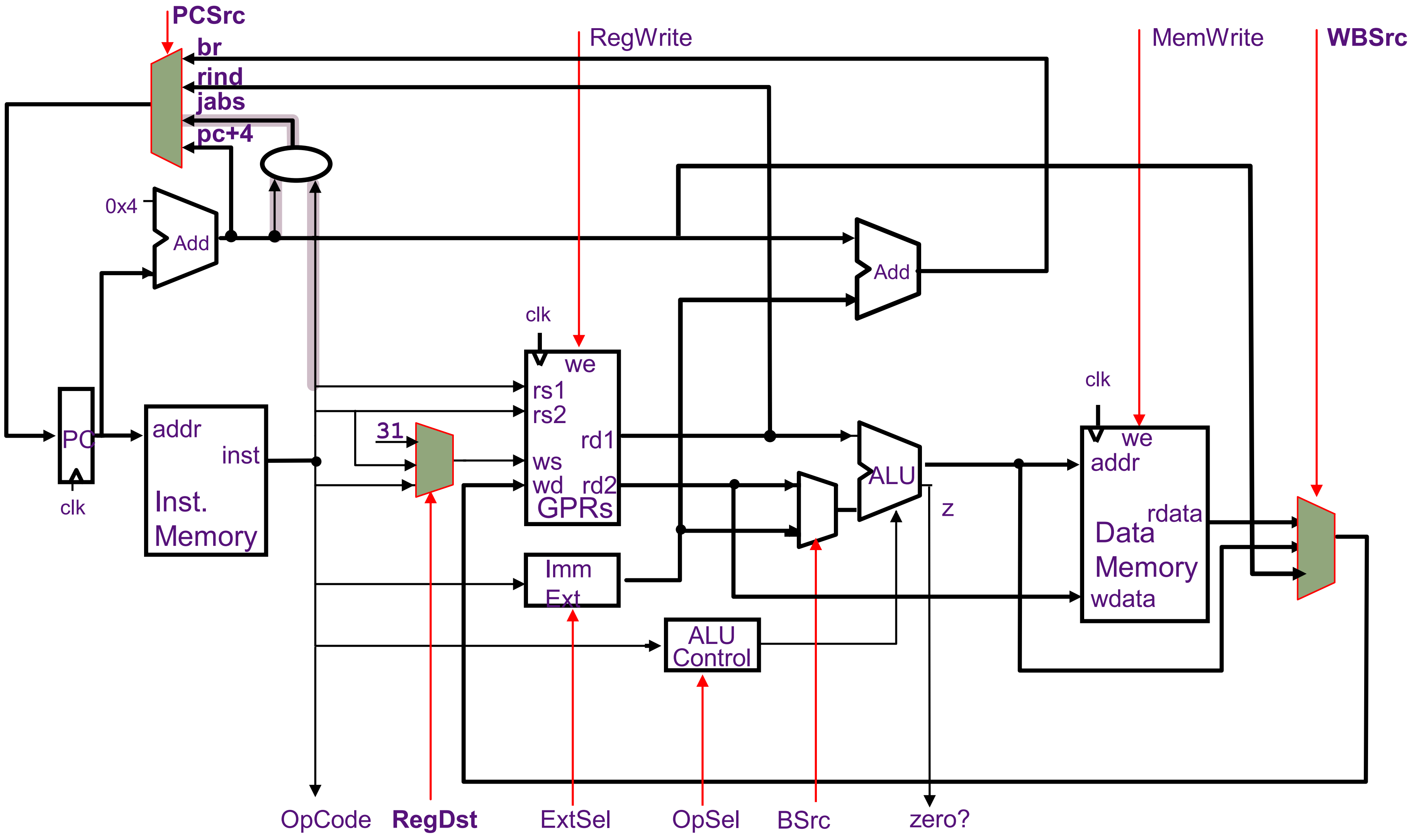
Absolute Jumps (J, JAL)



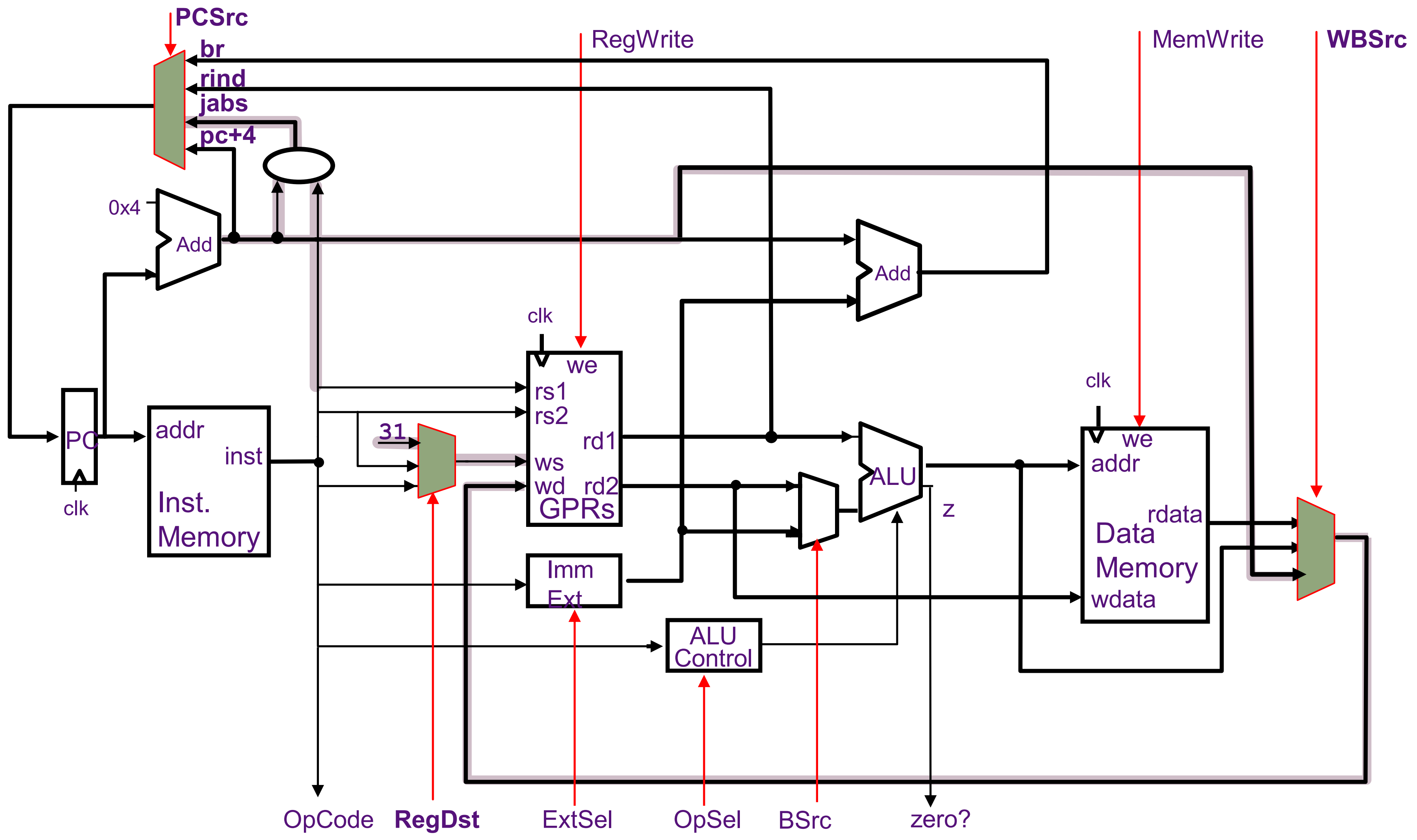
Absolute Jumps (J, JAL)



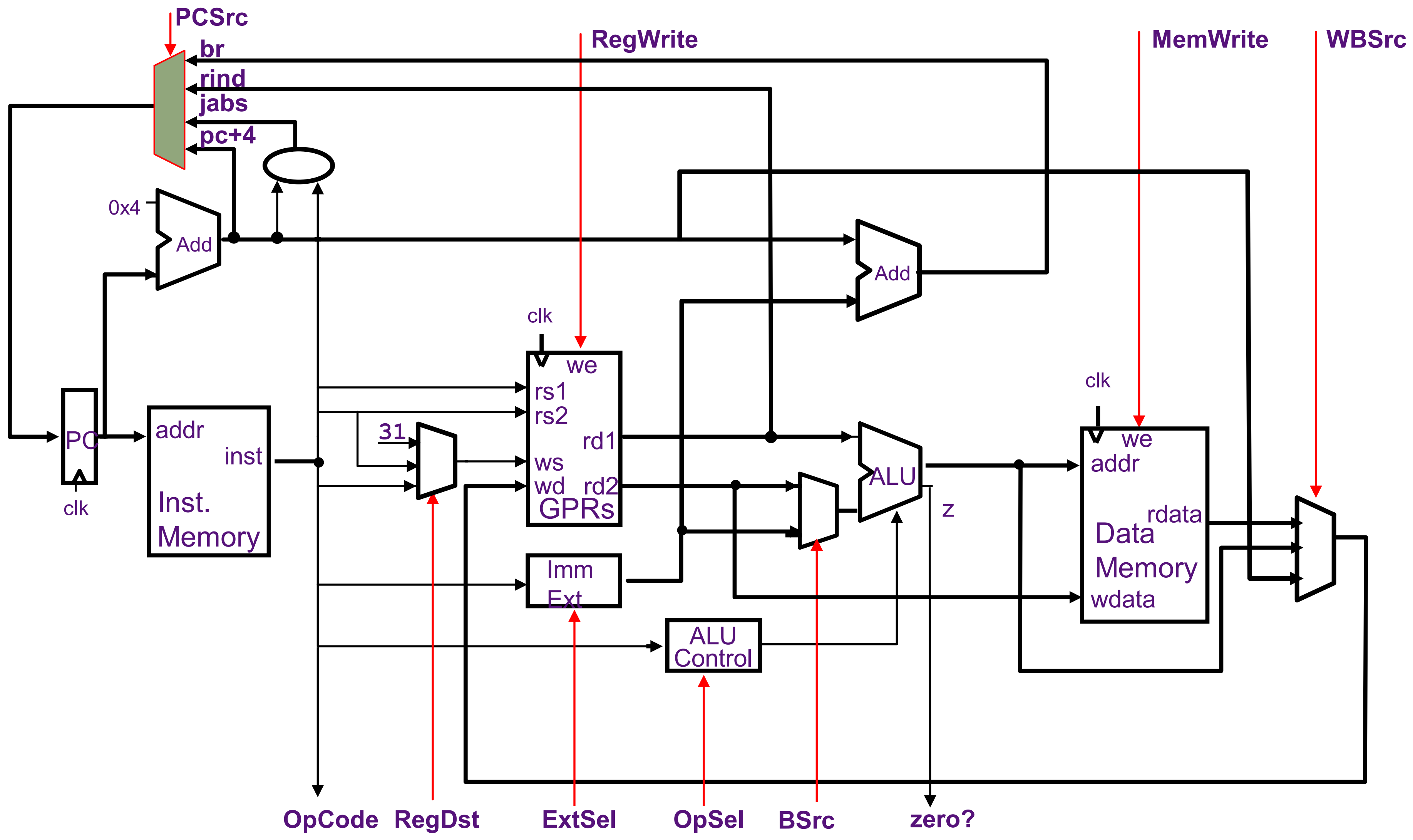
Absolute Jumps (J, JAL)



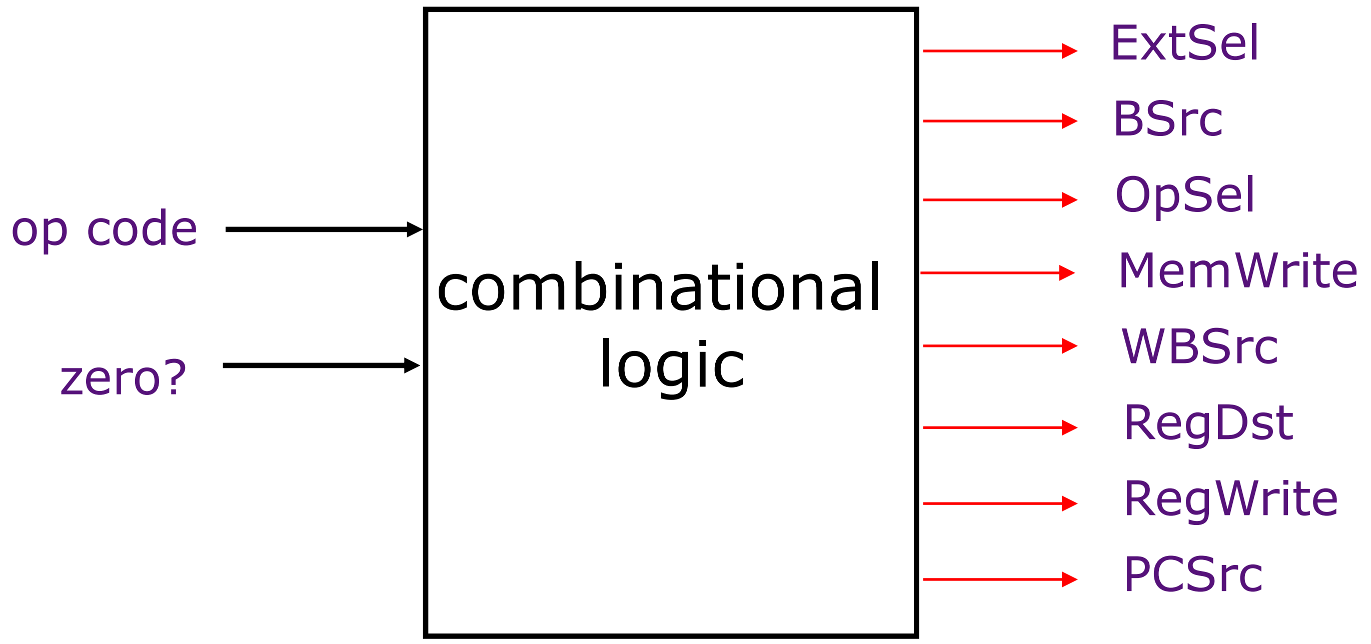
Absolute Jumps (J, JAL)



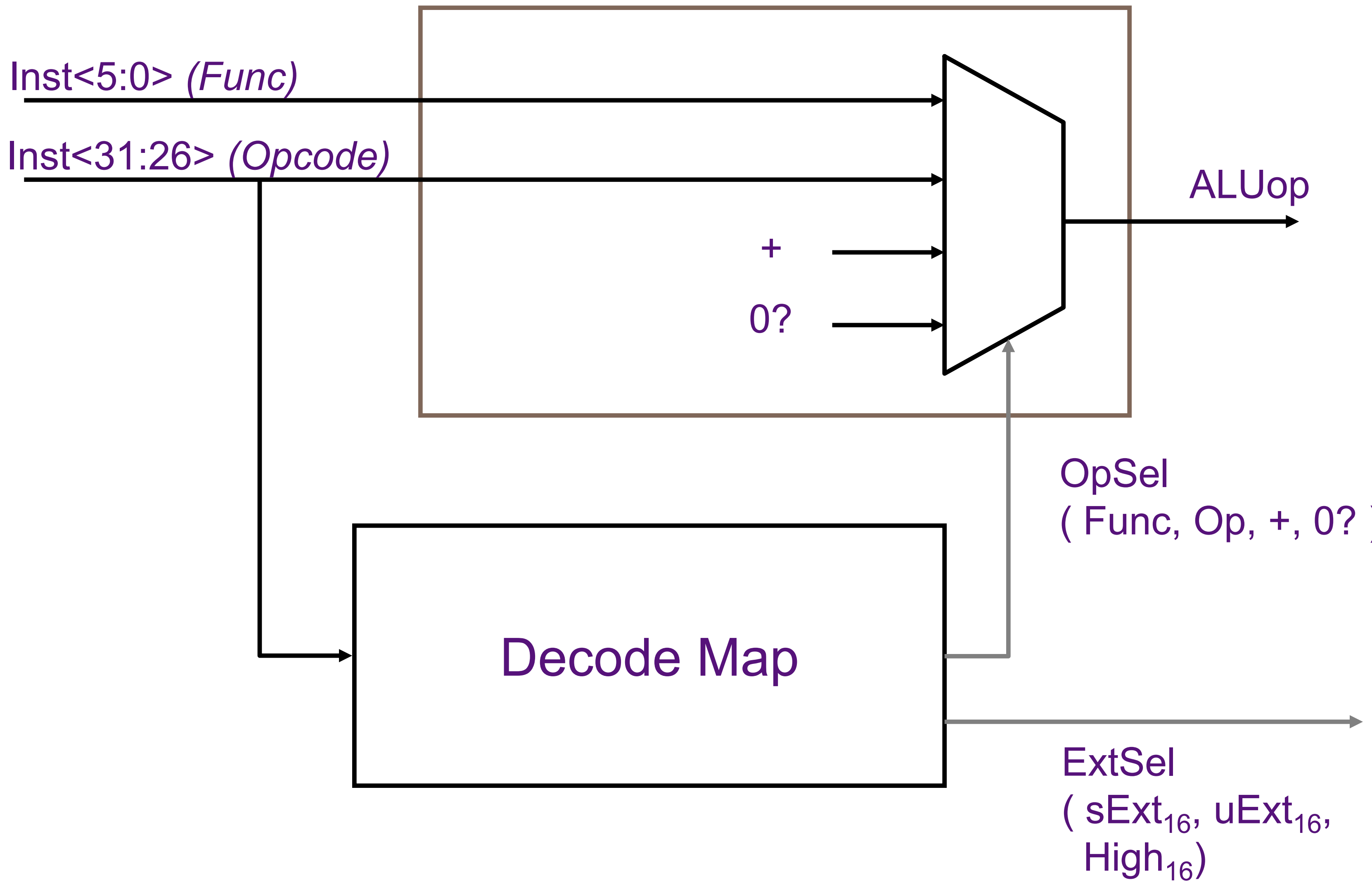
Harvard-Style Datapath for MIPS



Hardwired Control is pure Combinational Logic



ALU Control & Immediate Extension



Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU								
ALUi								
ALUiu								
LW								
SW								
$\text{BEQZ}_{z=0}$								
$\text{BEQZ}_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*							
ALUi								
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg						
ALUi								
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func					
ALUi								
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no				
ALUi								
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no	yes			
ALUi								
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no	yes	ALU		
ALUi								
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no	yes	ALU	rd	
ALUi								
ALUiu								
LW								
SW								
$\text{BEQZ}_{z=0}$								
$\text{BEQZ}_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no	yes	ALU	rd	pc+4
ALUi								
ALUiu								
LW								
SW								
$\text{BEQZ}_{z=0}$								
$\text{BEQZ}_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

Hardwired Control Table

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	no	yes	ALU	rd	pc+4
ALUi	$sExt_{16}$							
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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						
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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					
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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		pc+4
ALUiu								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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								
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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	$uExt_{16}$							
LW								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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								
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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	$sExt_{16}$							
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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						
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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	+					
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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				
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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			
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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		
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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	
SW								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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								
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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
$BEQZ_{z=0}$								
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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
$BEQZ_{z=0}$	$sExt_{16}$							
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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
$BEQZ_{z=0}$	$sExt_{16}$	*						
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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?					
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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	*	*	
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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
$BEQZ_{z=1}$								
J								
JAL								
JR								
JALR								

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

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

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	*	*	
J								
JAL								
JR								
JALR								

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

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

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								
JAL								
JR								
JALR								

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

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

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	*	*	*					
JAL								
JR								
JALR								

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

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

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	*	*	
JAL								
JR								
JALR								

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

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

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								
JR								
JALR								

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

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

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				
JR								
JALR								

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

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

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			
JR								
JALR								

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

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

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		
JR								
JALR								

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

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

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	
JR								
JALR								

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

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

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
$\text{BEQZ}_{z=0}$	sExt_{16}	*	0?	no	no	*	*	br
$\text{BEQZ}_{z=1}$	sExt_{16}	*	0?	no	no	*	*	pc+4
J	*	*	*	no	no	*	*	jabs
JAL	*	*	*	no	yes	PC	R31	jabs
JR								
JALR								

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

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

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	*	*	*					
JALR								

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

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

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	*	*	
JALR								

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

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

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								

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

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

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
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?

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

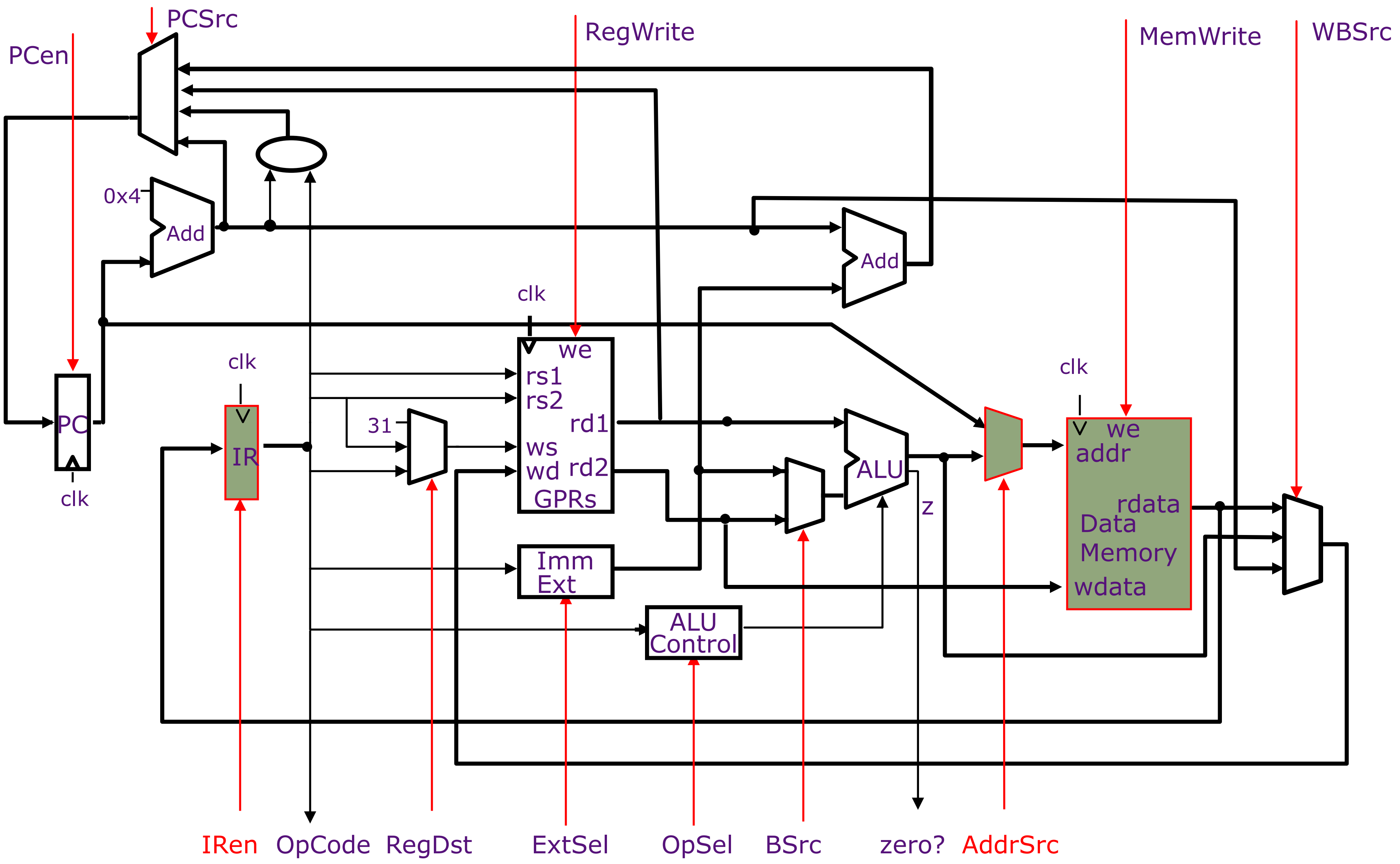
Princeton challenge

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

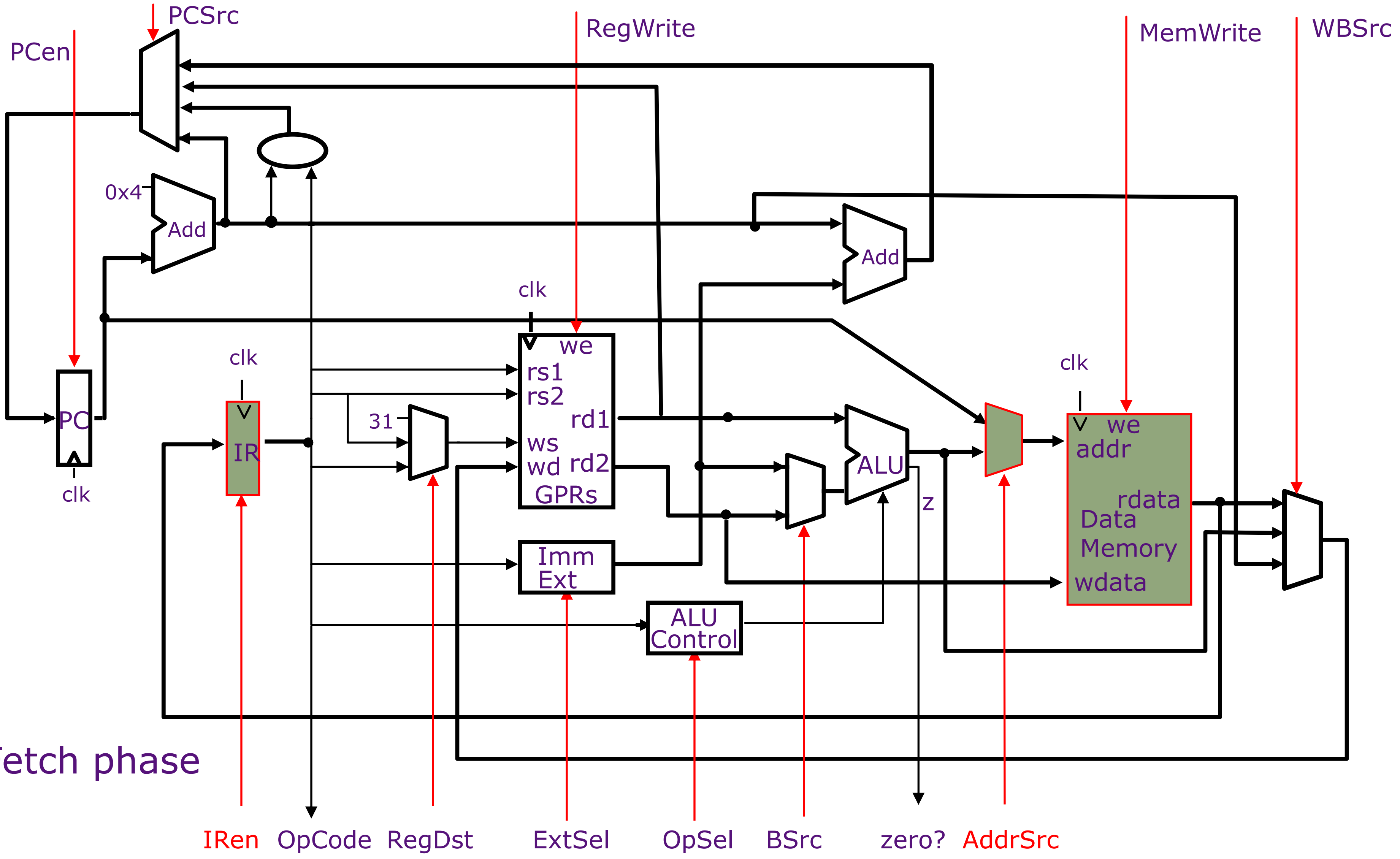
At least the instruction fetch and a Load (or Store) cannot be executed in the same cycle

Structural hazard

Princeton Microarchitecture
Datapath & Control

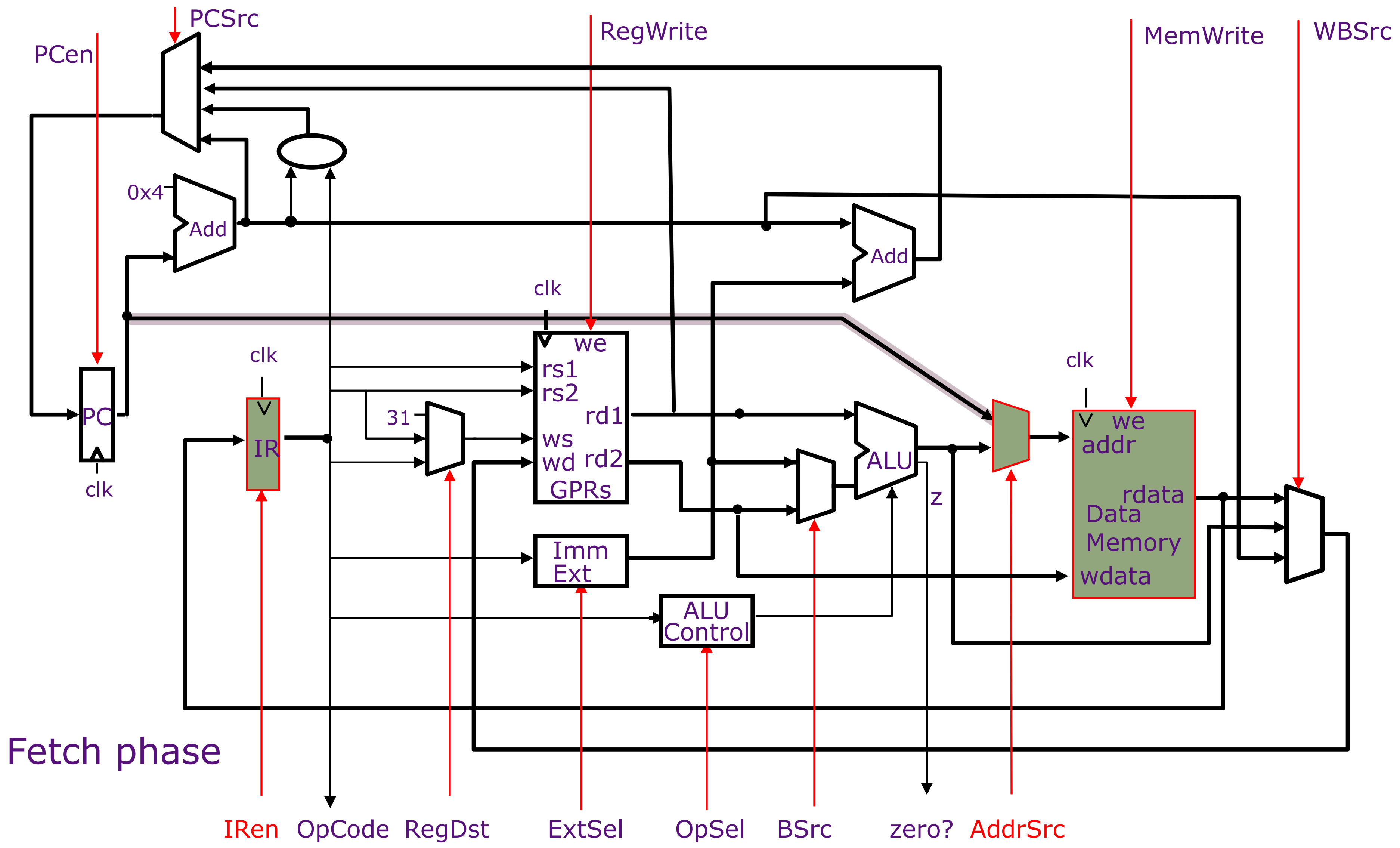


Princeton Microarchitecture
Datapath & Control



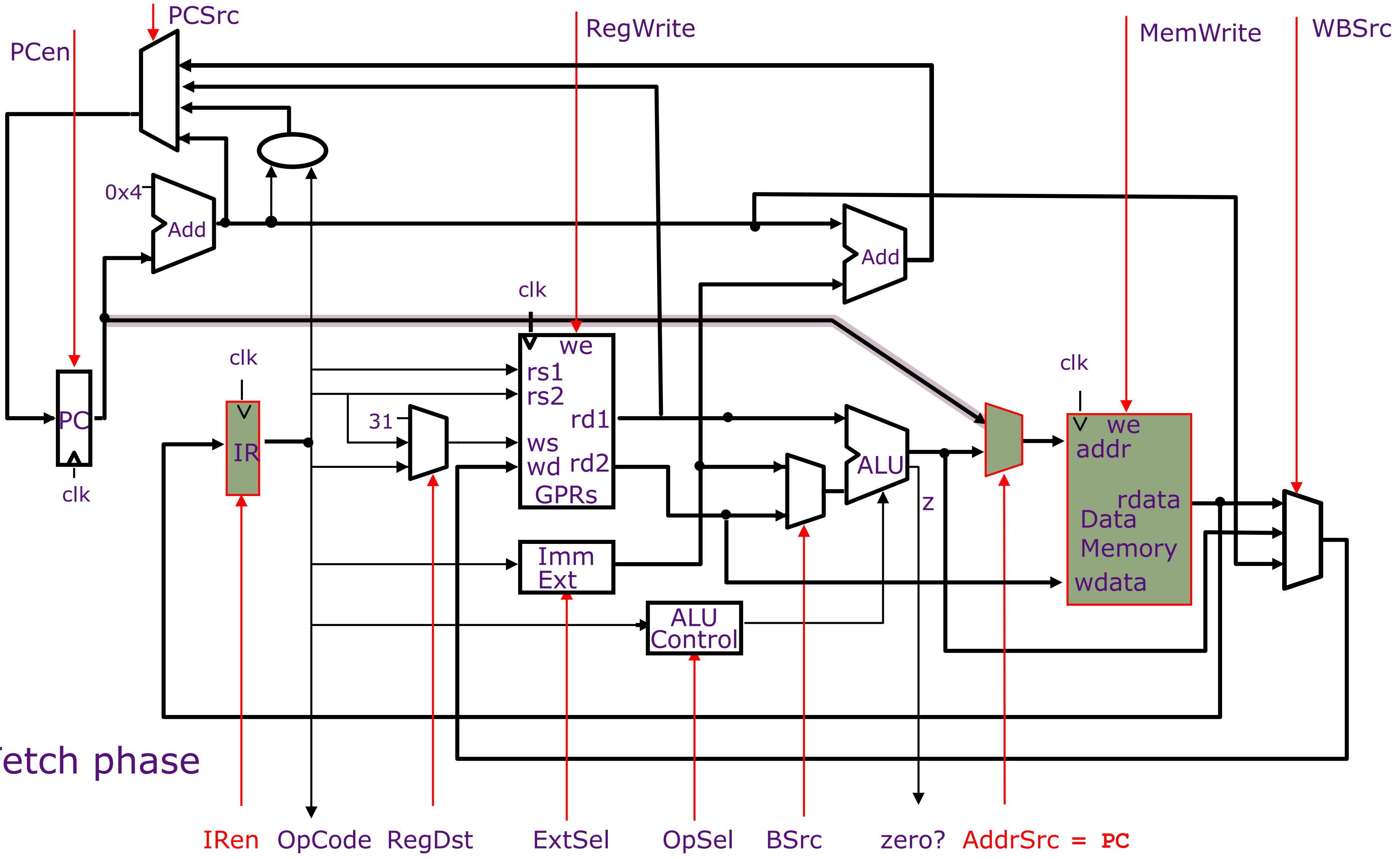
Fetch phase

Princeton Microarchitecture
Datapath & Control



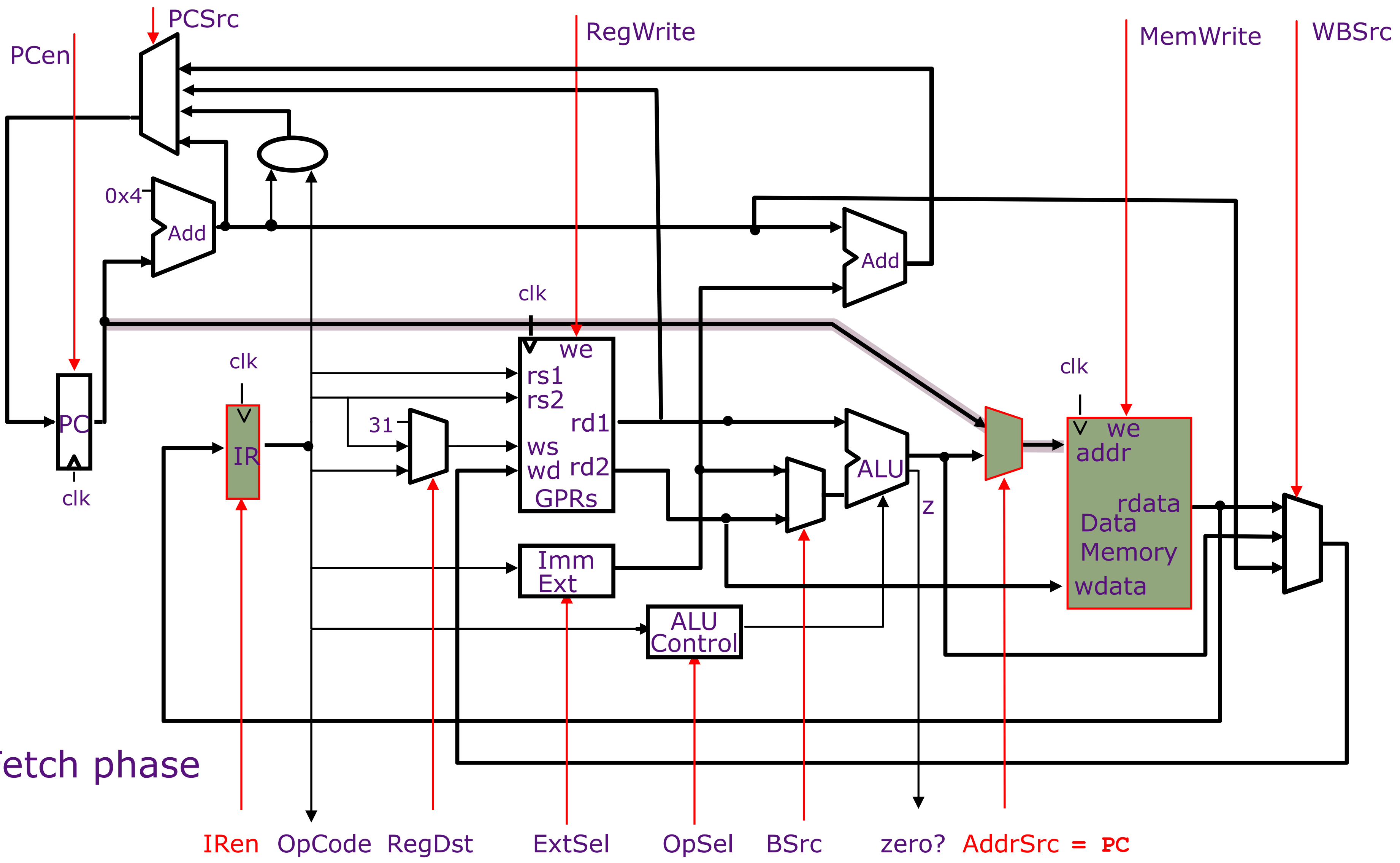
Fetch phase

Princeton Microarchitecture
Datapath & Control



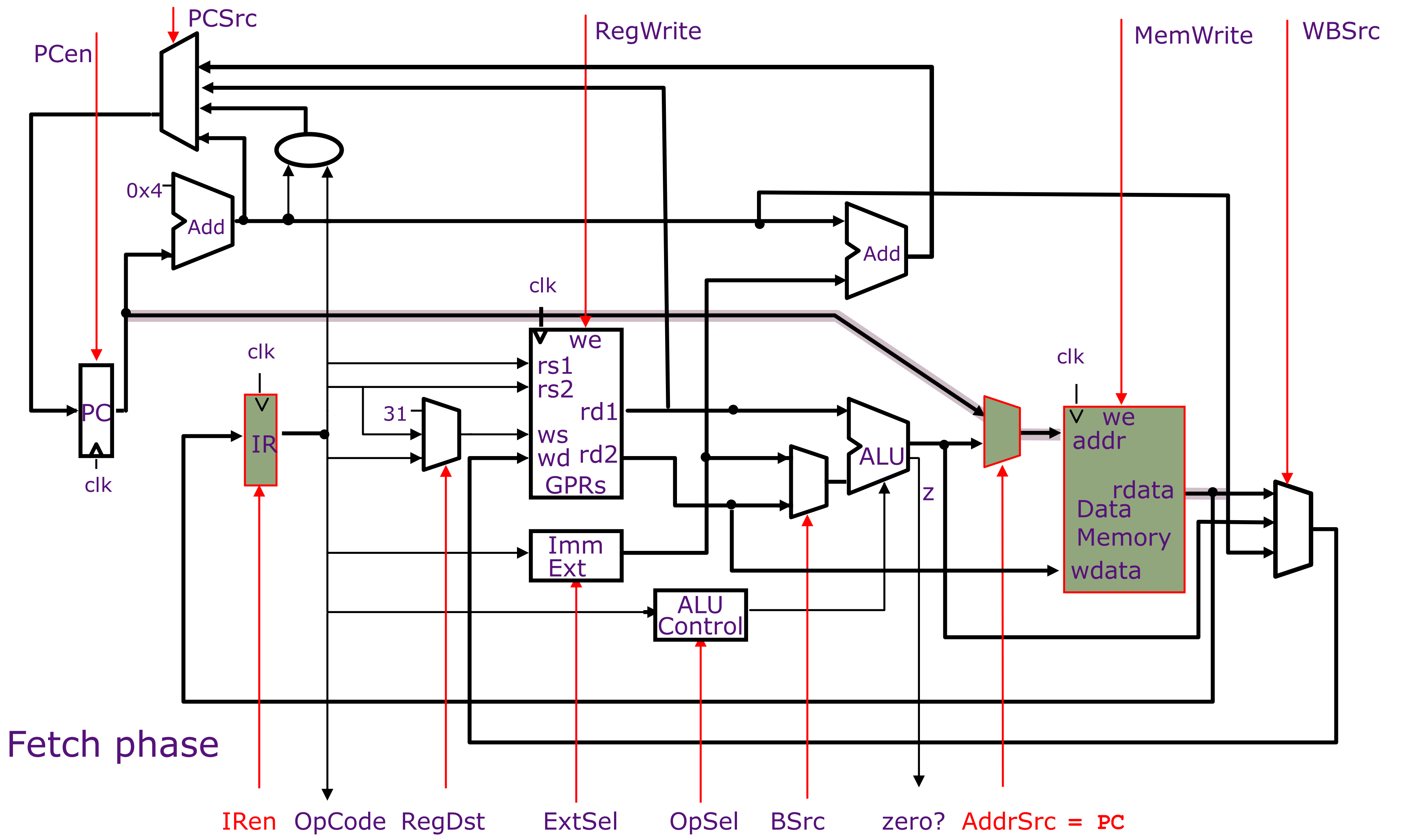
Fetch phase

Princeton Microarchitecture
Datapath & Control



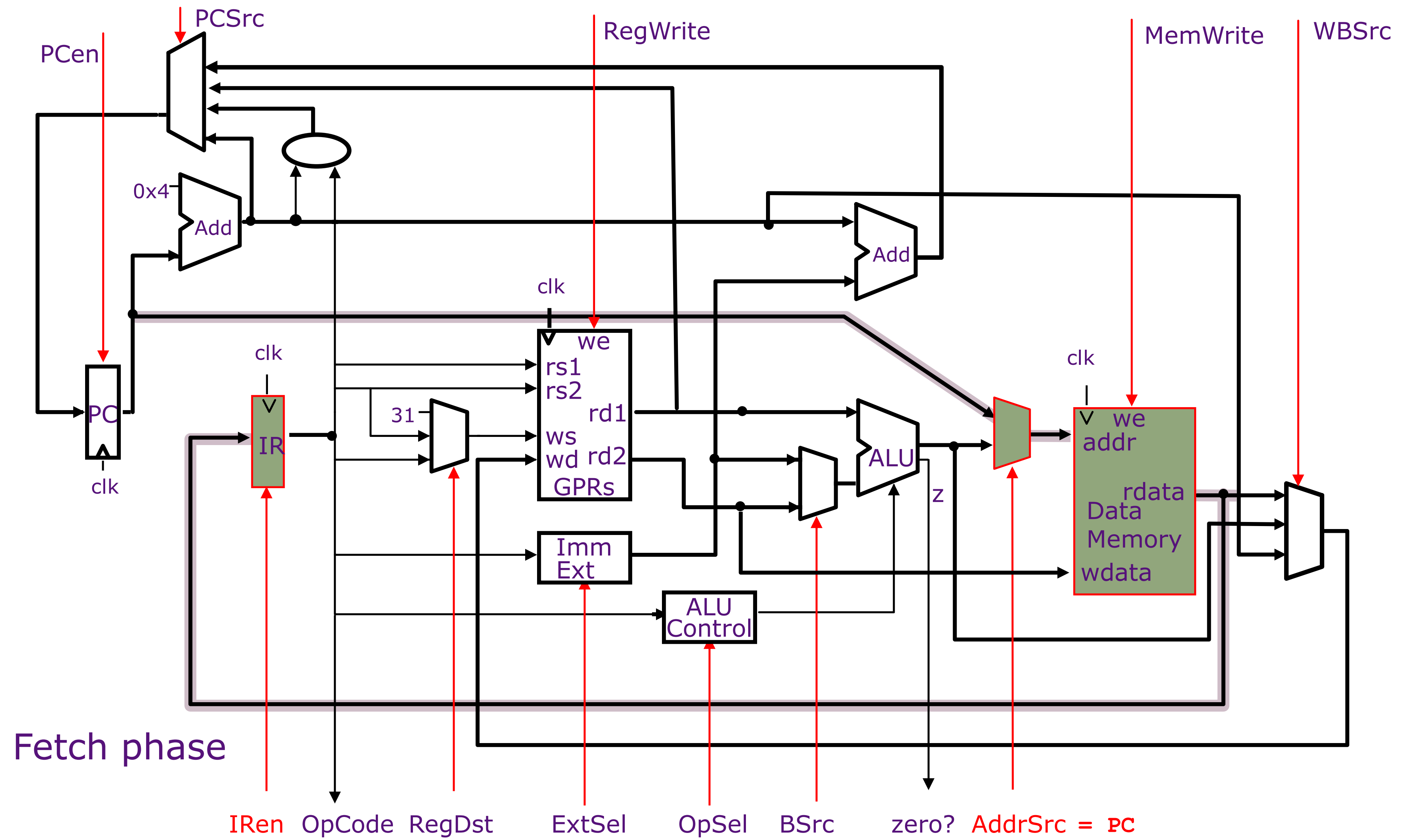
Fetch phase

Princeton Microarchitecture
Datapath & Control



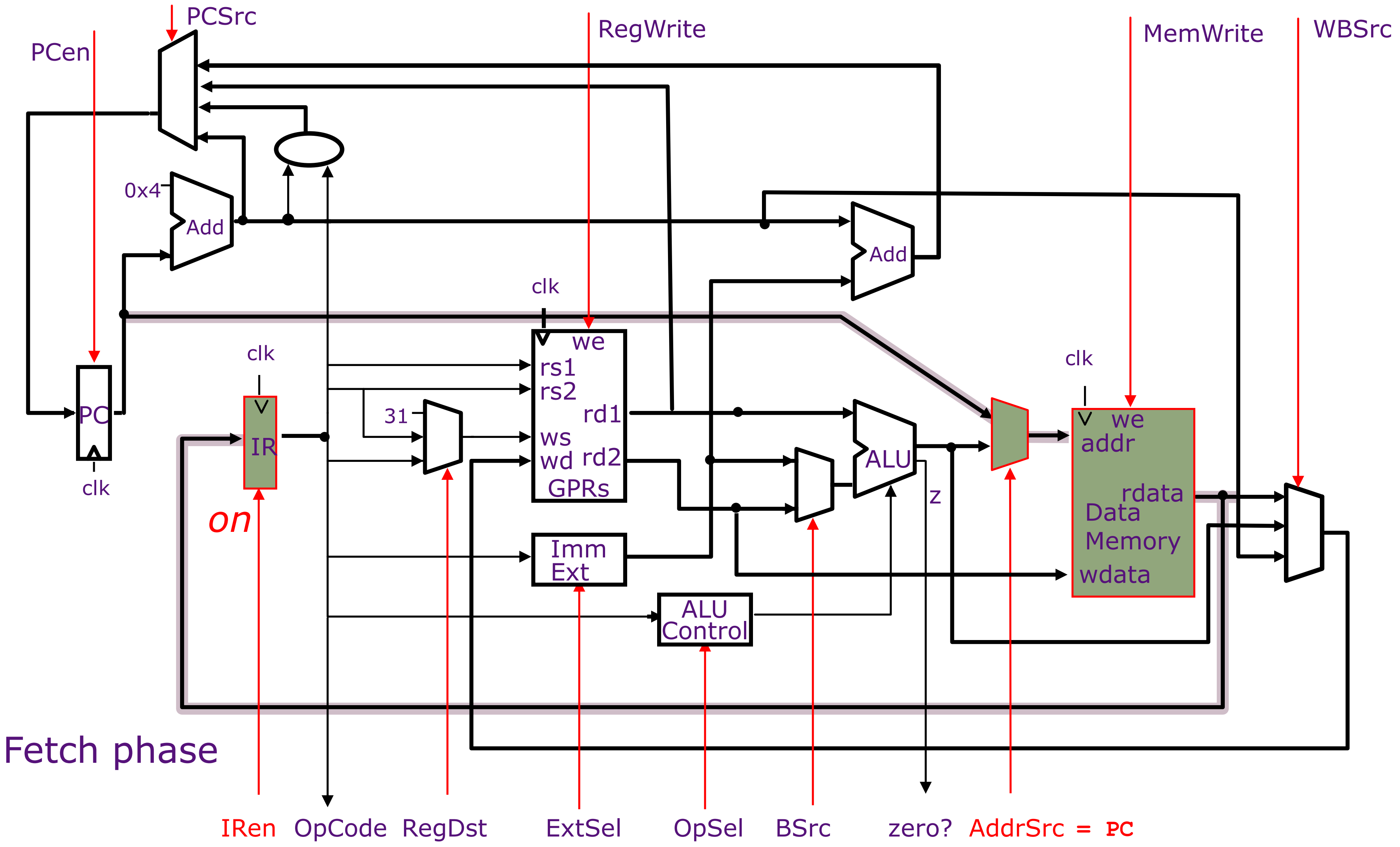
Fetch phase

Princeton Microarchitecture
Datapath & Control



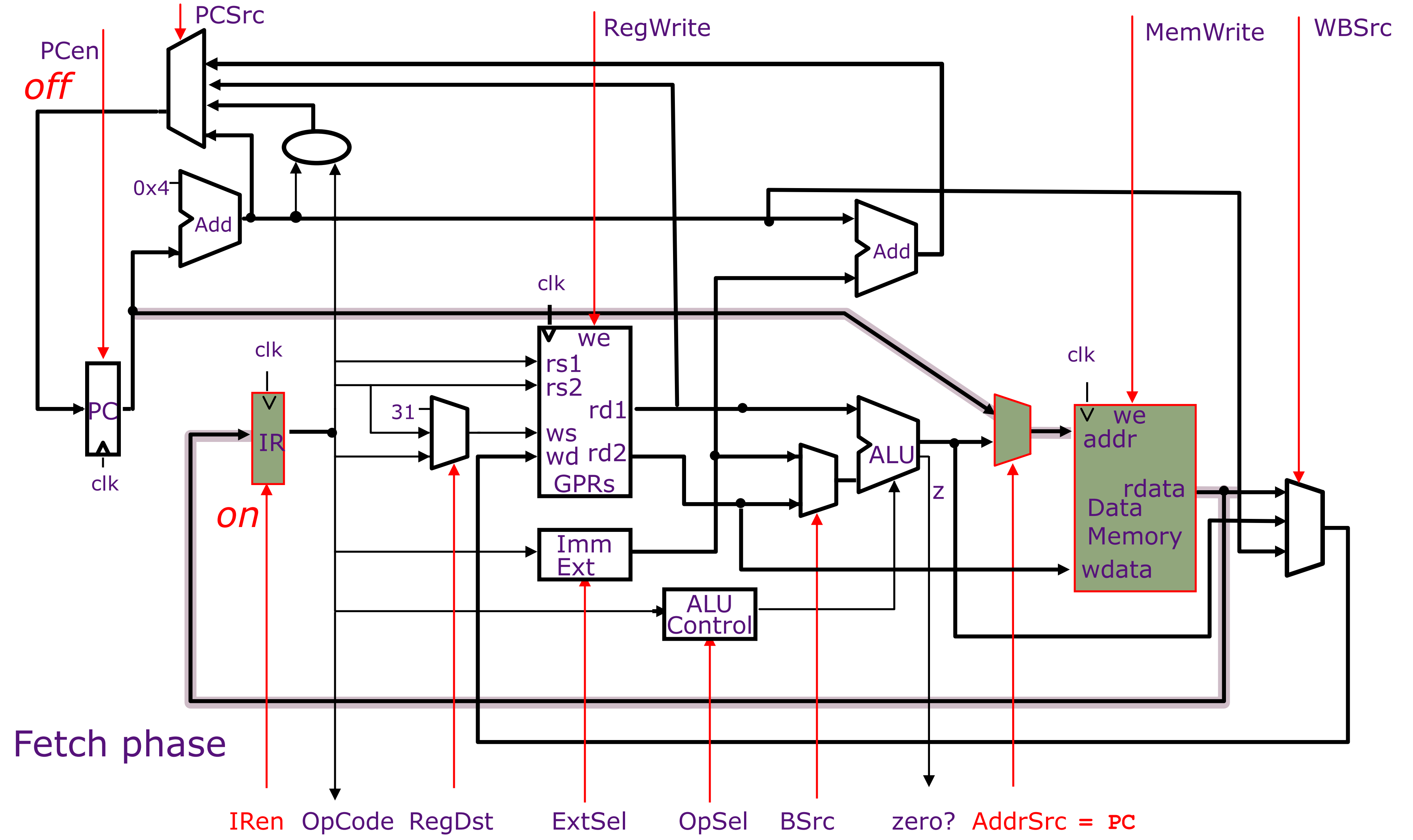
Fetch phase

Princeton Microarchitecture
Datapath & Control



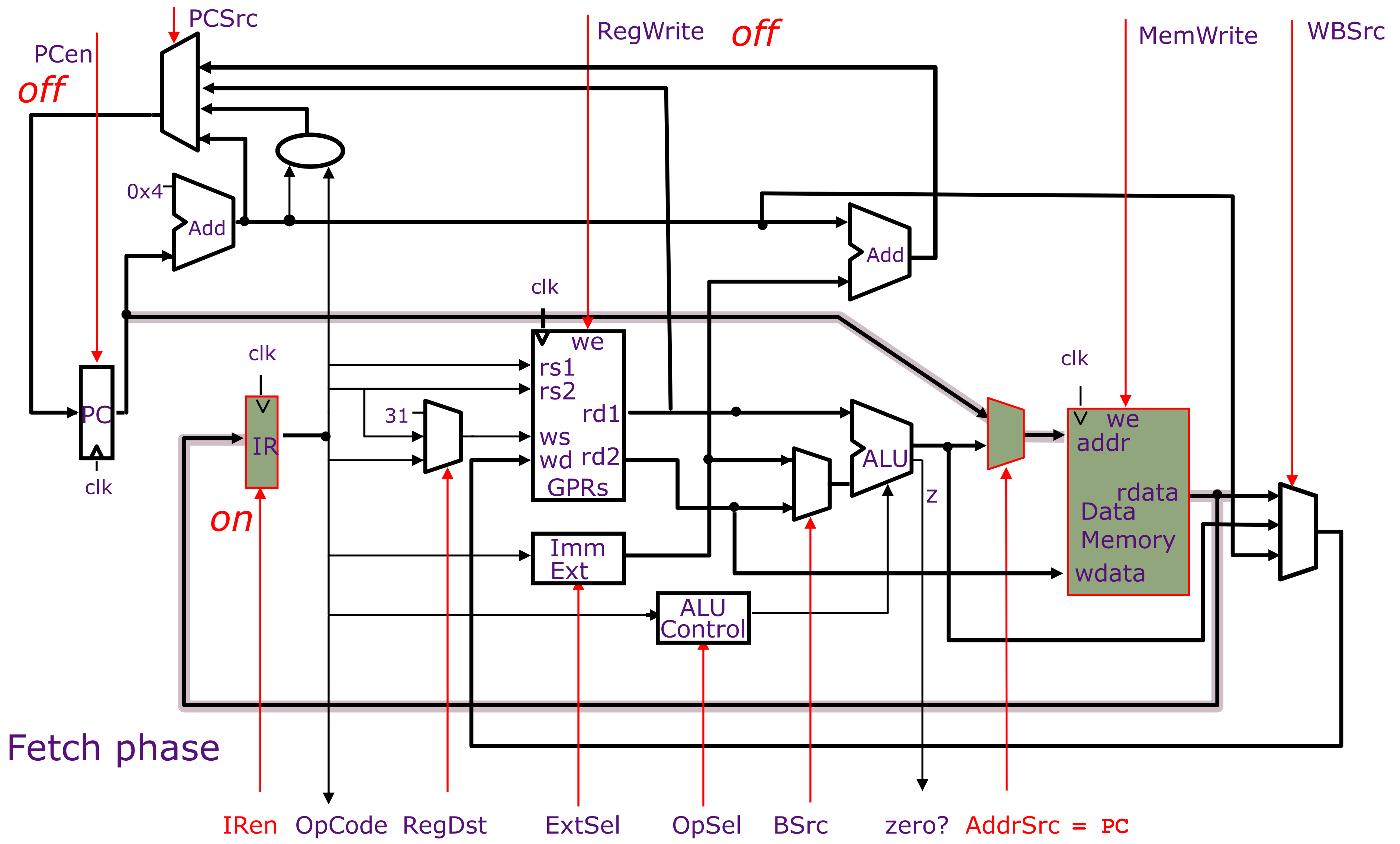
Fetch phase

Princeton Microarchitecture
Datapath & Control



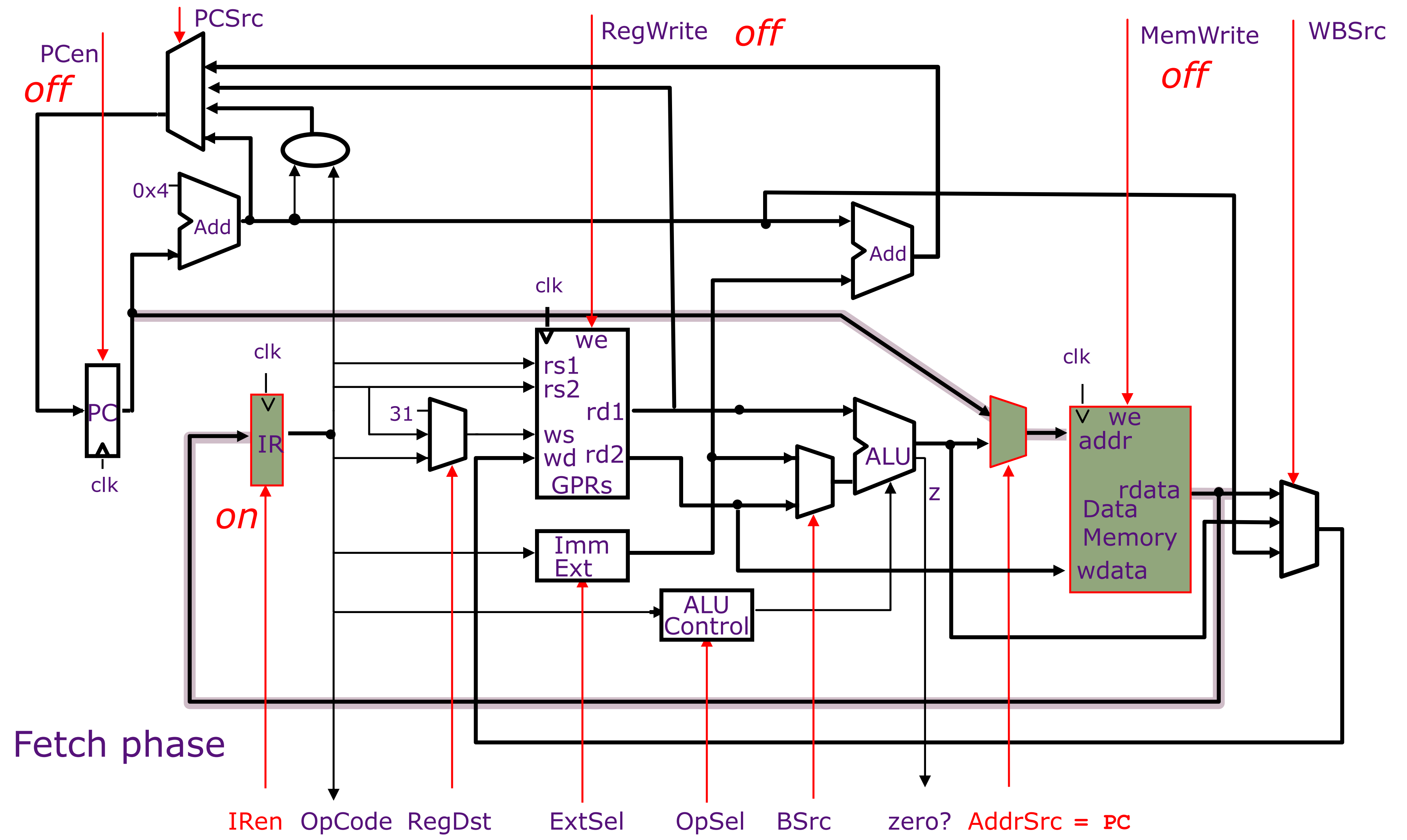
Fetch phase

Princeton Microarchitecture
Datapath & Control



Fetch phase

Princeton Microarchitecture
Datapath & Control

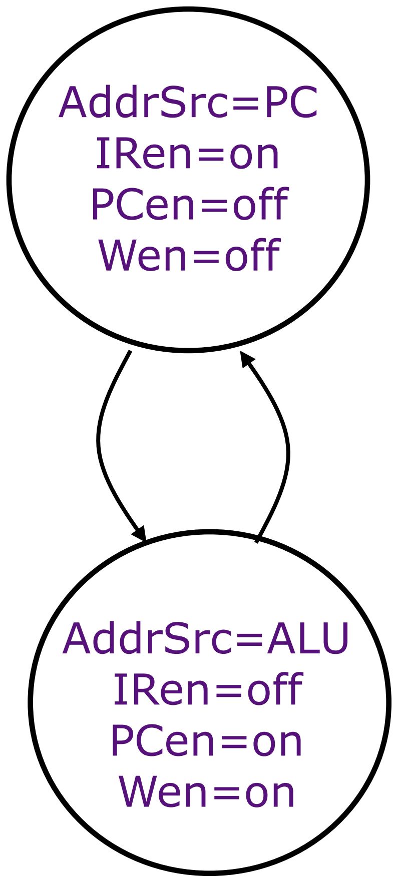


Fetch phase

Two-State Controller:
Princeton Architecture

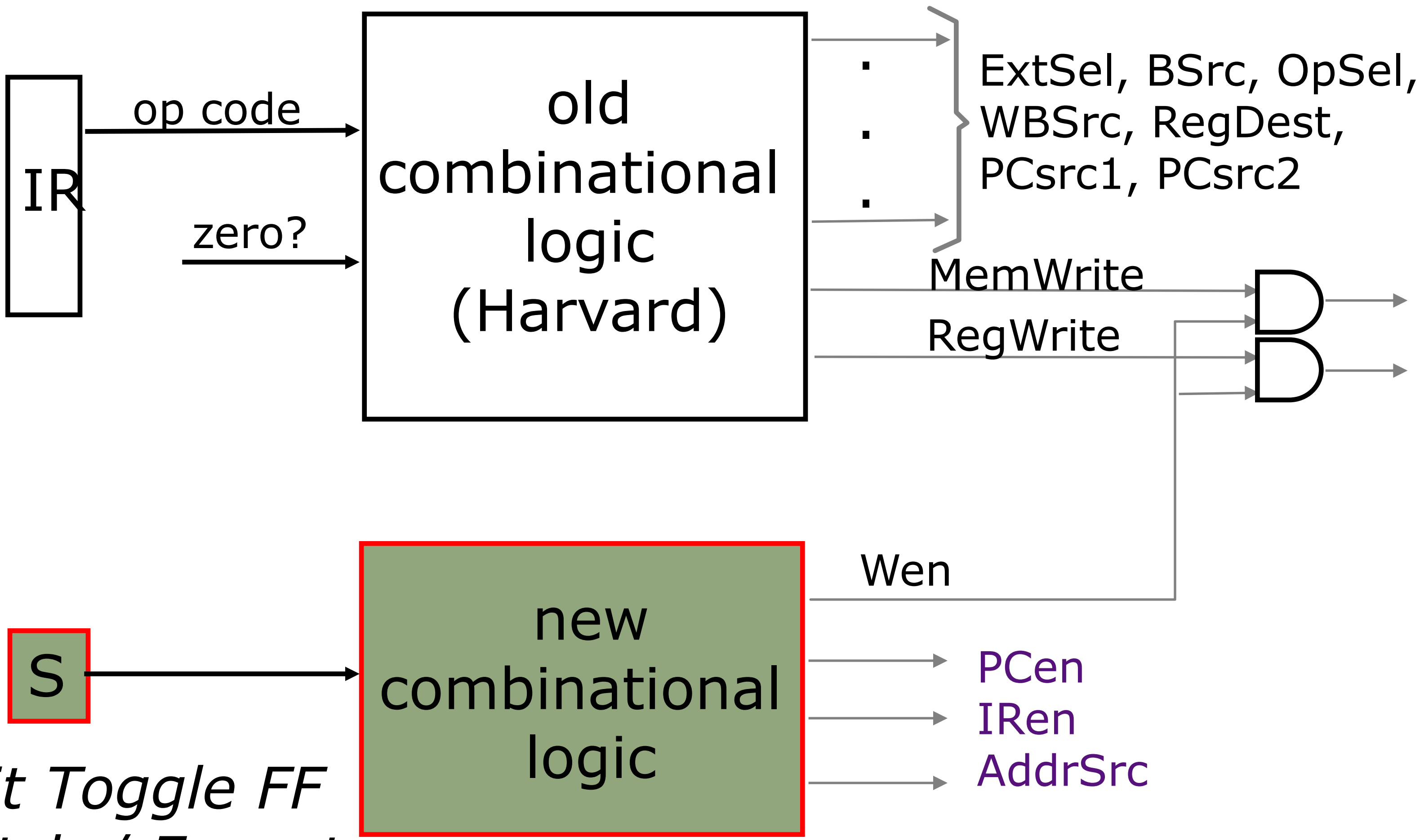
fetch phase

execute phase



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
Wen
S
new combinational logic
PCen
IRen
AddrSrc

1-bit Toggle FF
I-fetch / Execute

Clock Period

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

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

while in the hardwired Harvard architecture

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

which will execute instructions faster?

Clock Rate vs CPI

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 ?

Stay tuned!

CPI = Clock cycles Per Instruction