CSAIL

Single-Cycle Processors: Datapath & Control

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

Based on the material prepared by
Arvind and Krste Asanovic

Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
  - Defines set of programmer visible state
  - Defines instruction format (bit encoding) and instruction semantics
  - Examples: *MIPS, x86, IBM 360, JVM*
- Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z900 (c. 2001)
  - x86 implementations: *8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4 (c. 2000), AMD Athlon, 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

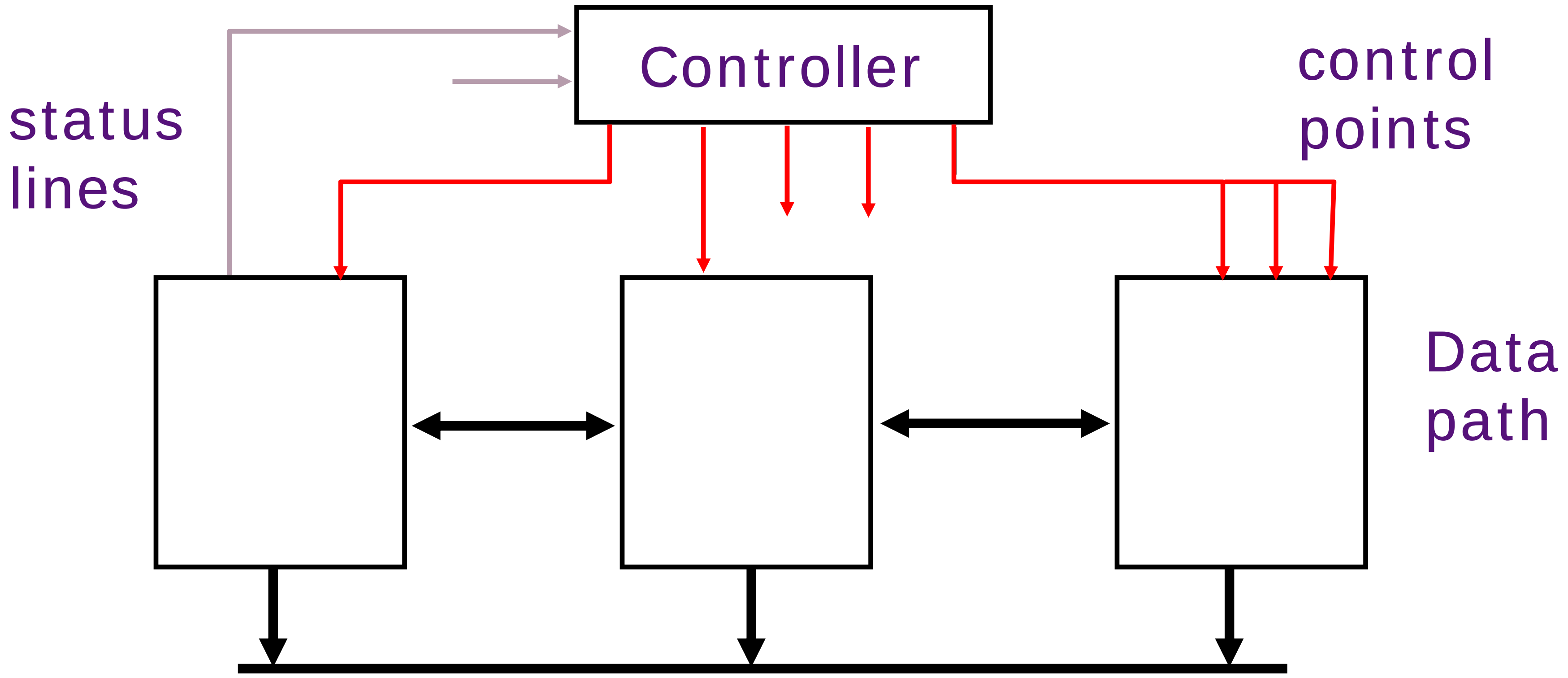
– Cycles per instructions (CPI) depends upon the ISA and the microarchitecture

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

this lecture $\longrightarrow$

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

Microarchitecture: *Implementation of an ISA*

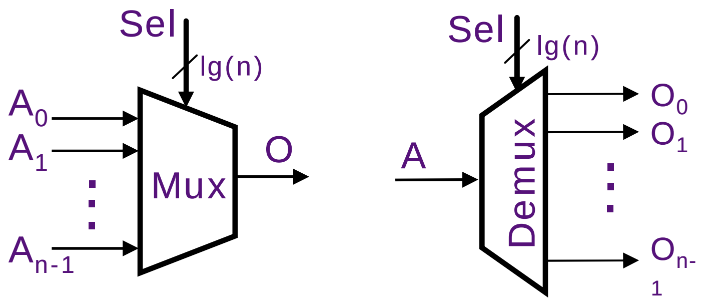


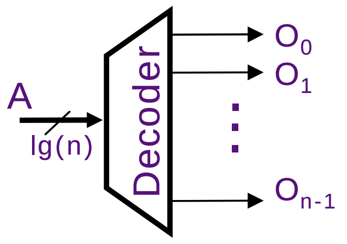
Structure: How components are connected.
Static

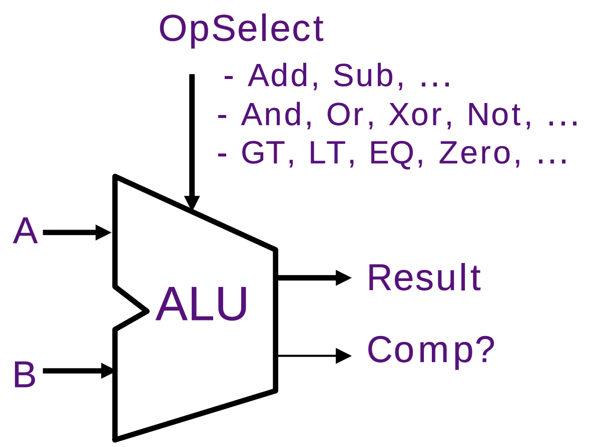
Behavior: How data moves between components
Dynamic

Hardware Elements

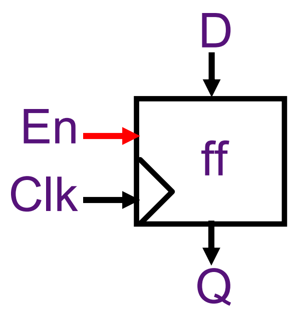
- Combinational circuits
  - Mux, Demux, Decoder, ALU, ...

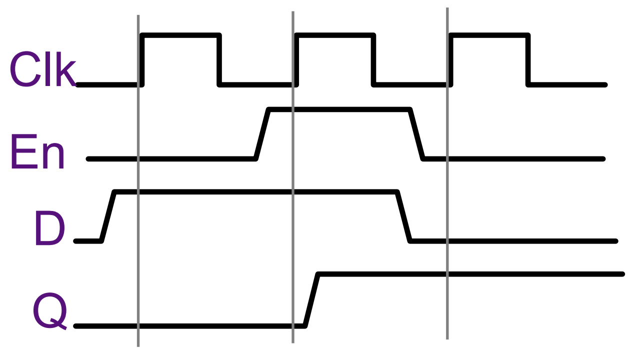


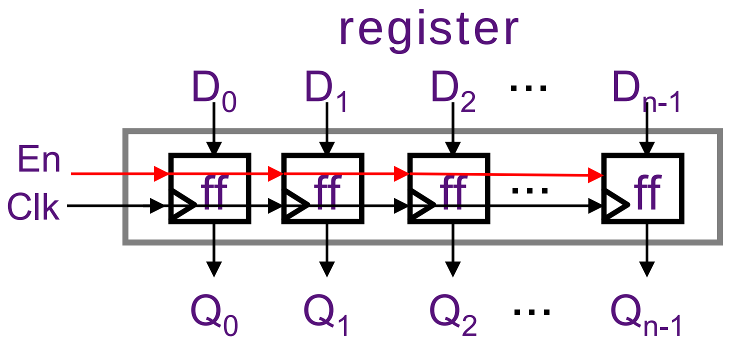




- Synchronous state elements
  - Flipflop, Register, Register file, SRAM, DRAM

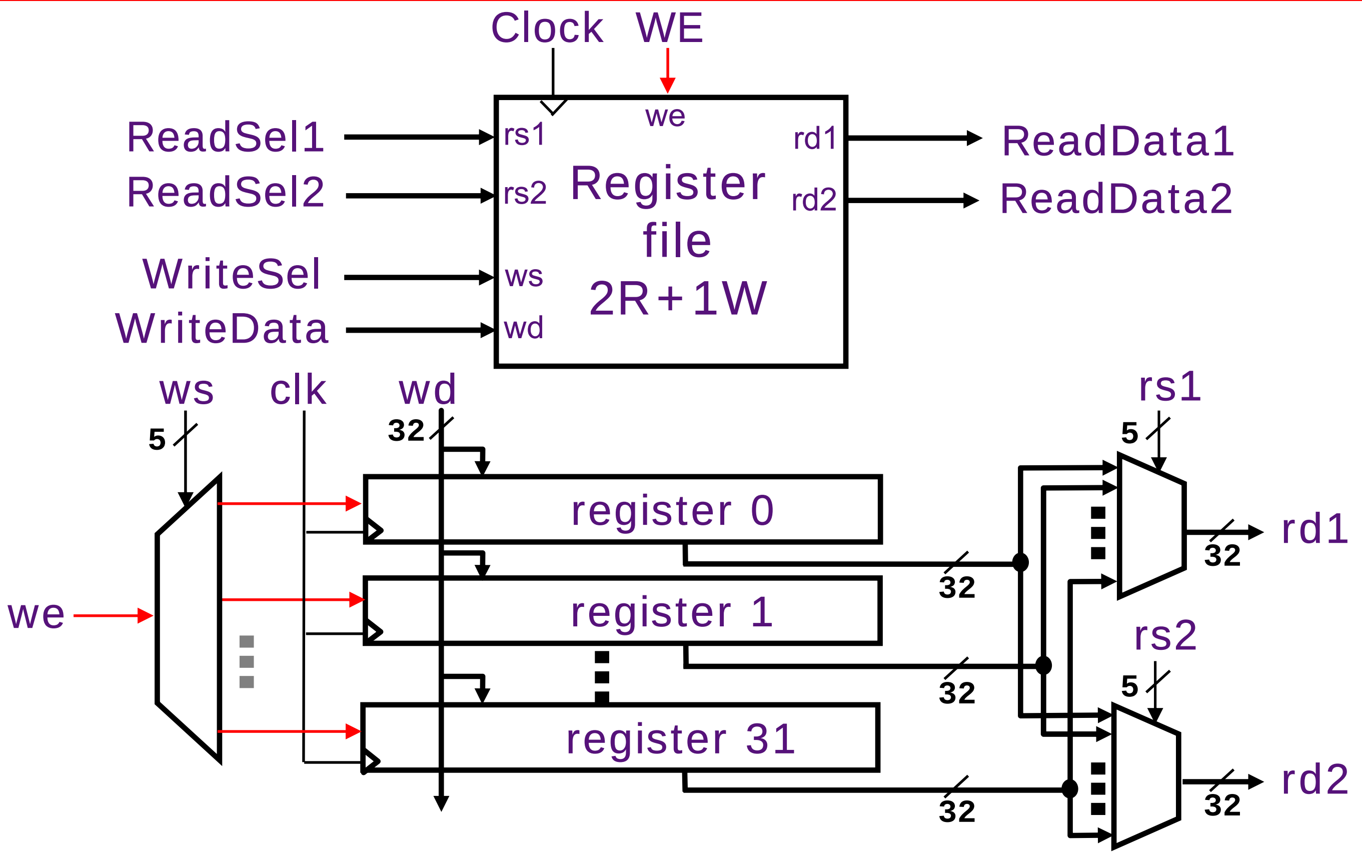






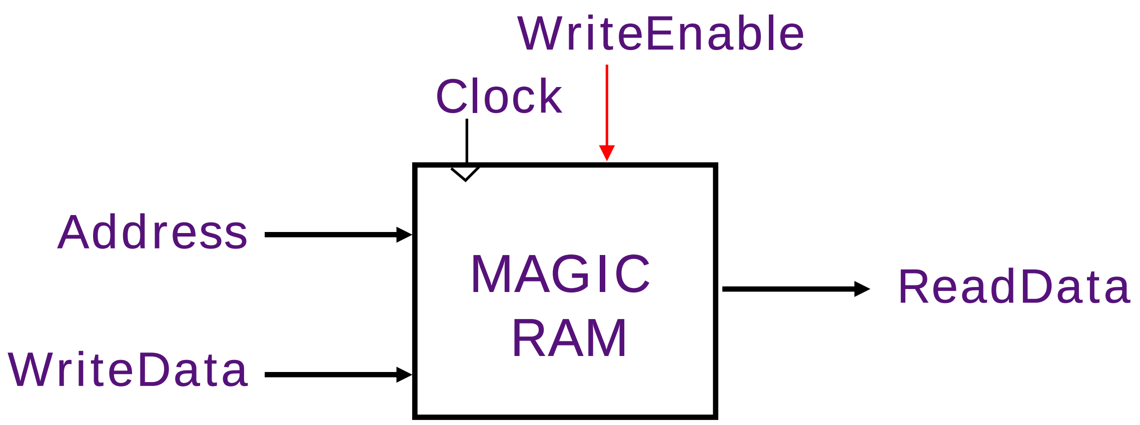
Edge-triggered: Data is sampled at the rising edge

Register Files



- No timing issues in reading a selected register
- Register files with a large number of ports are difficult to design
  - *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)
- a Write is performed at the rising clock edge if it is enabled

$\Rightarrow$ *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

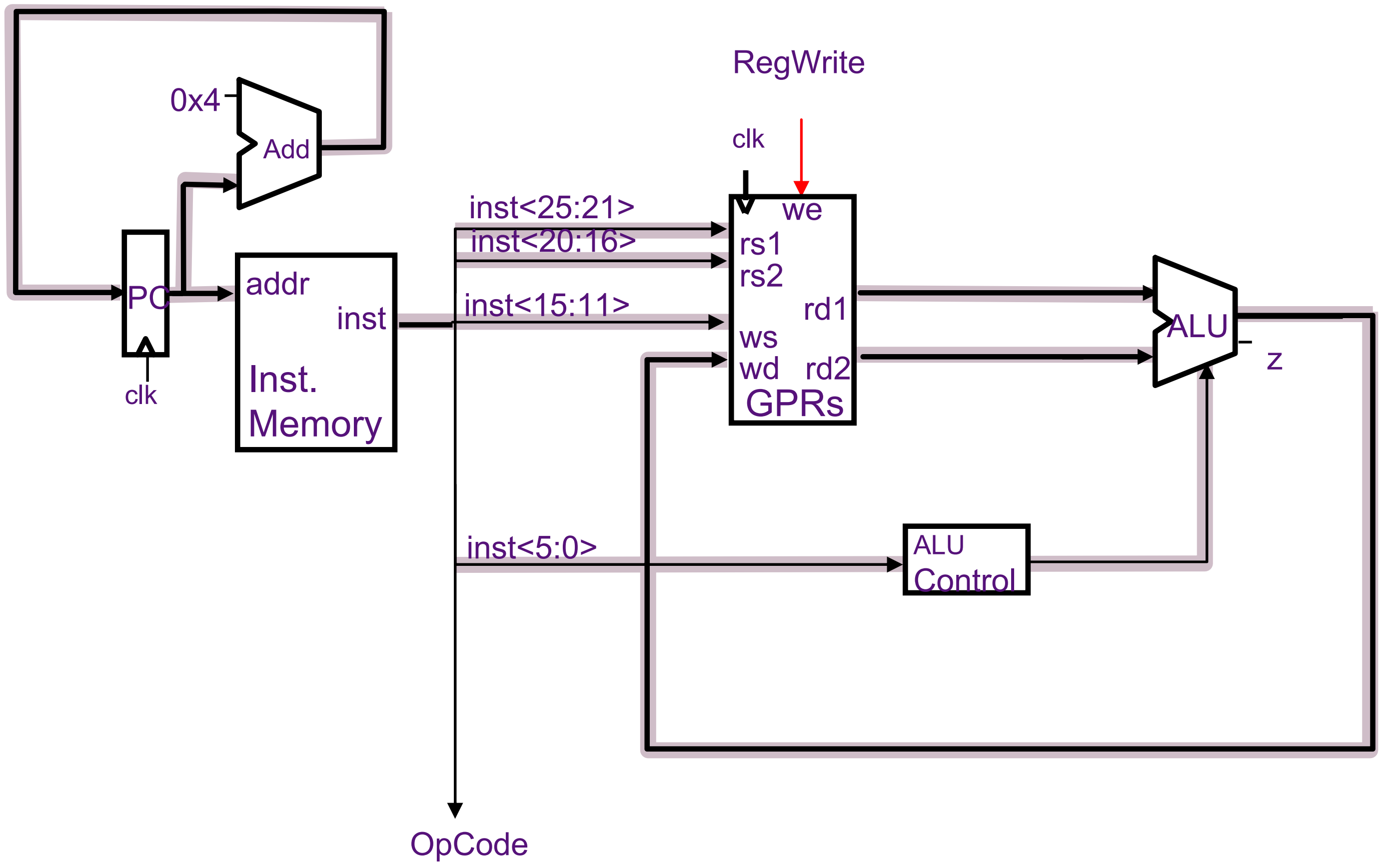
Instruction Execution

Execution of an instruction involves

1. instruction fetch
2. decode 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

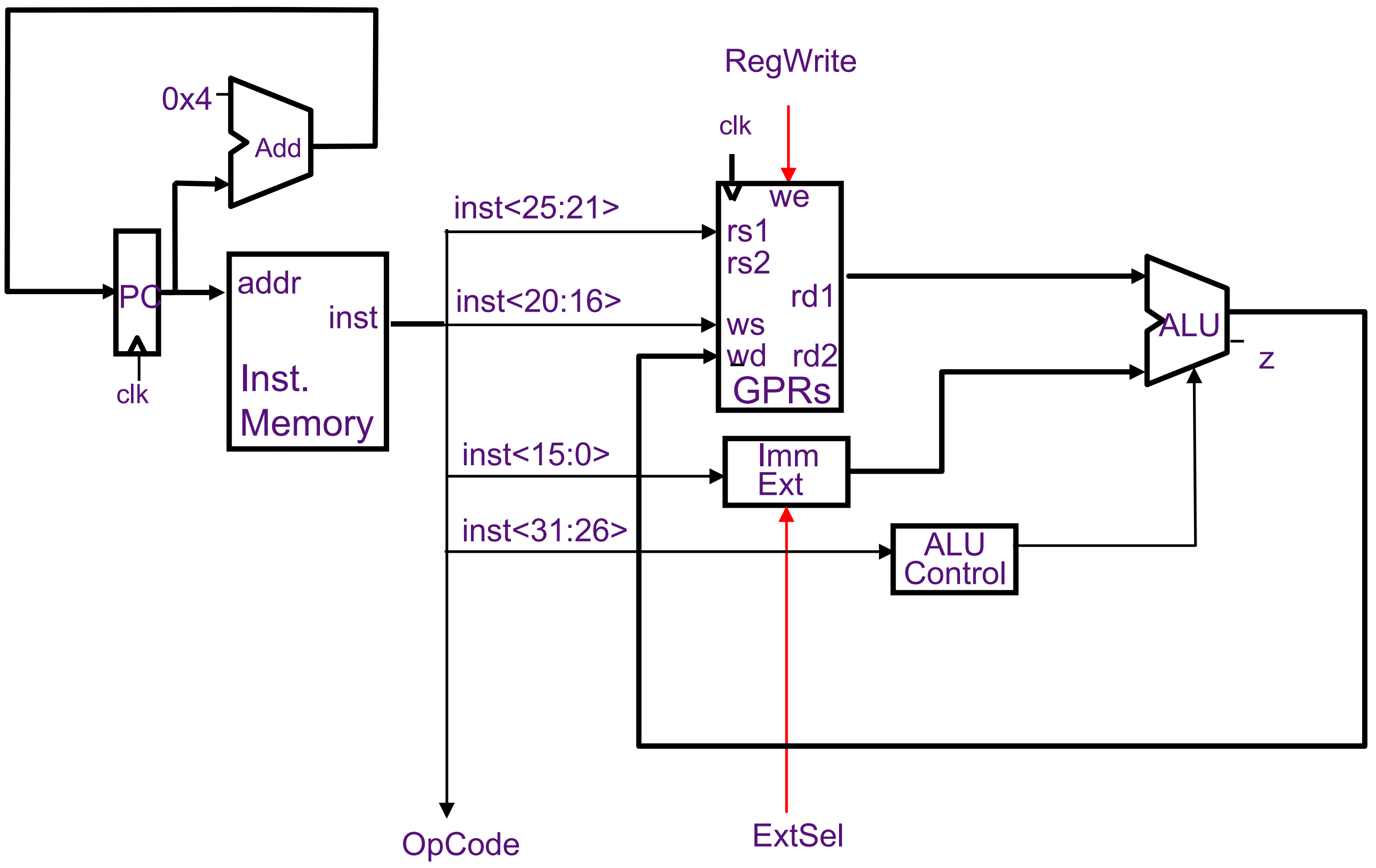


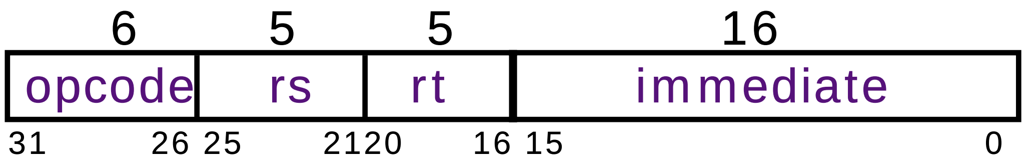
RegWrite Timing?

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)

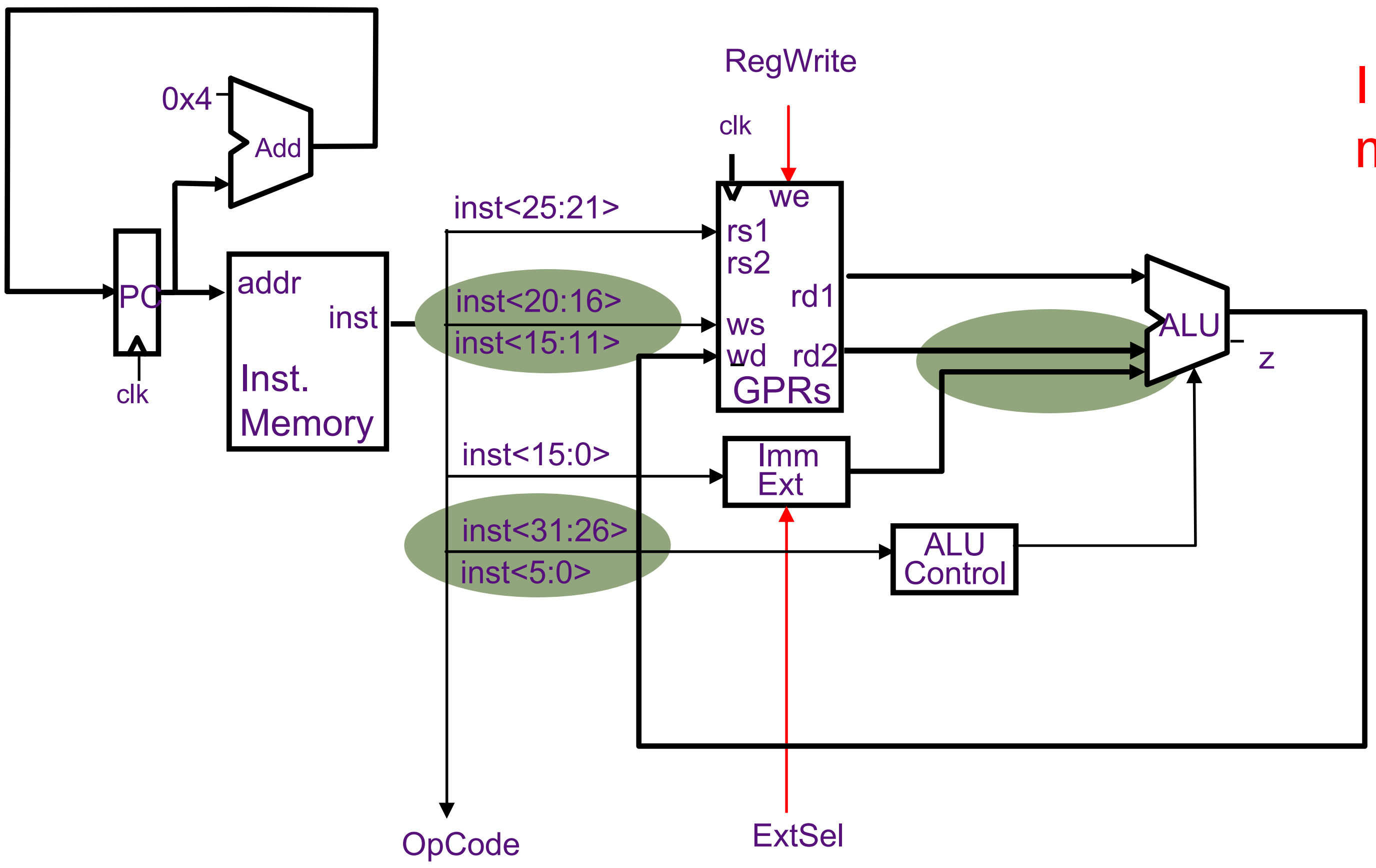
Datapath: Reg-Imm ALU Instructions





rt ← (rs) op immediate

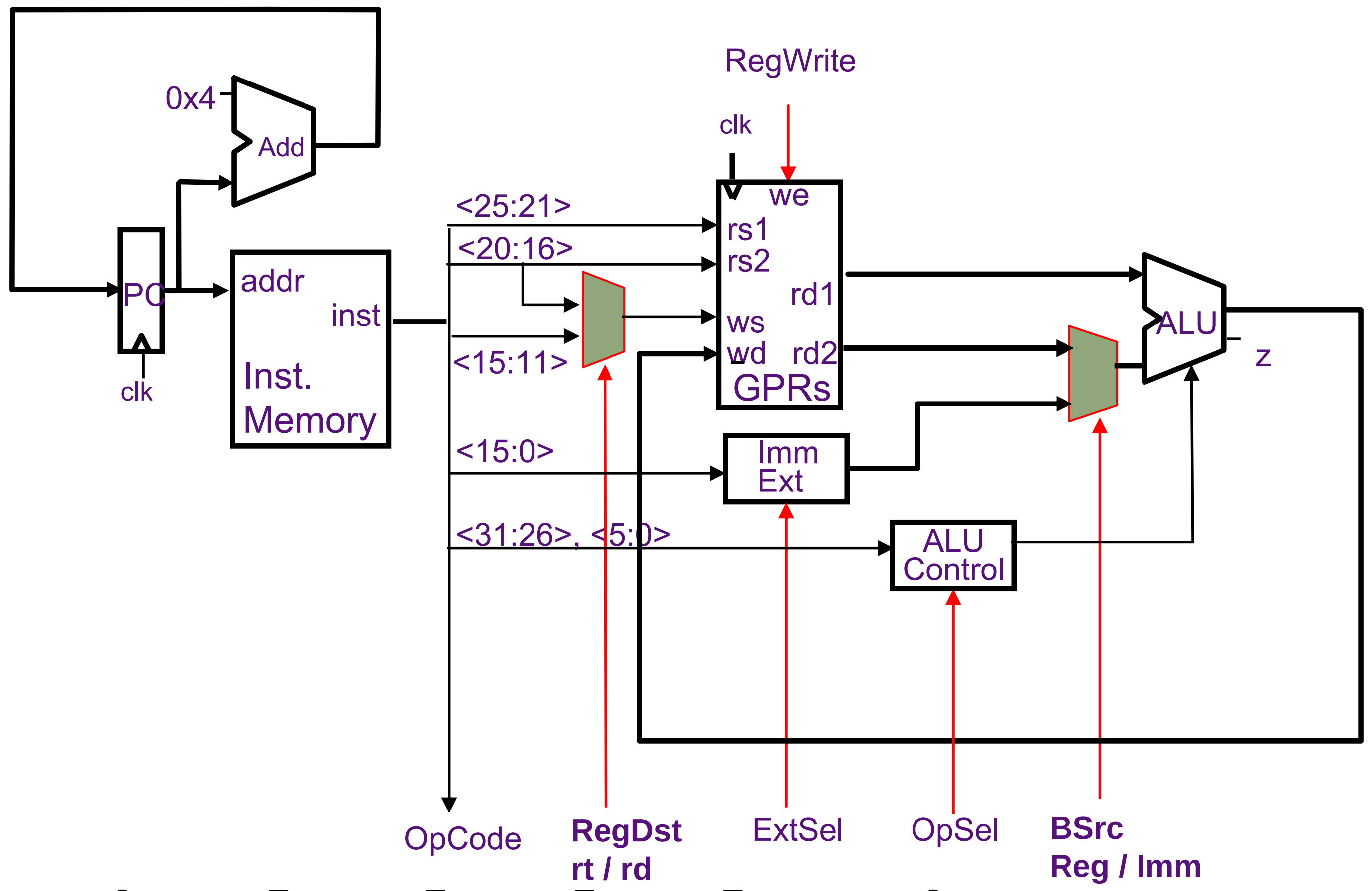
Conflicts in Merging Datapath



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 (Aiken and Mark 1 influence)

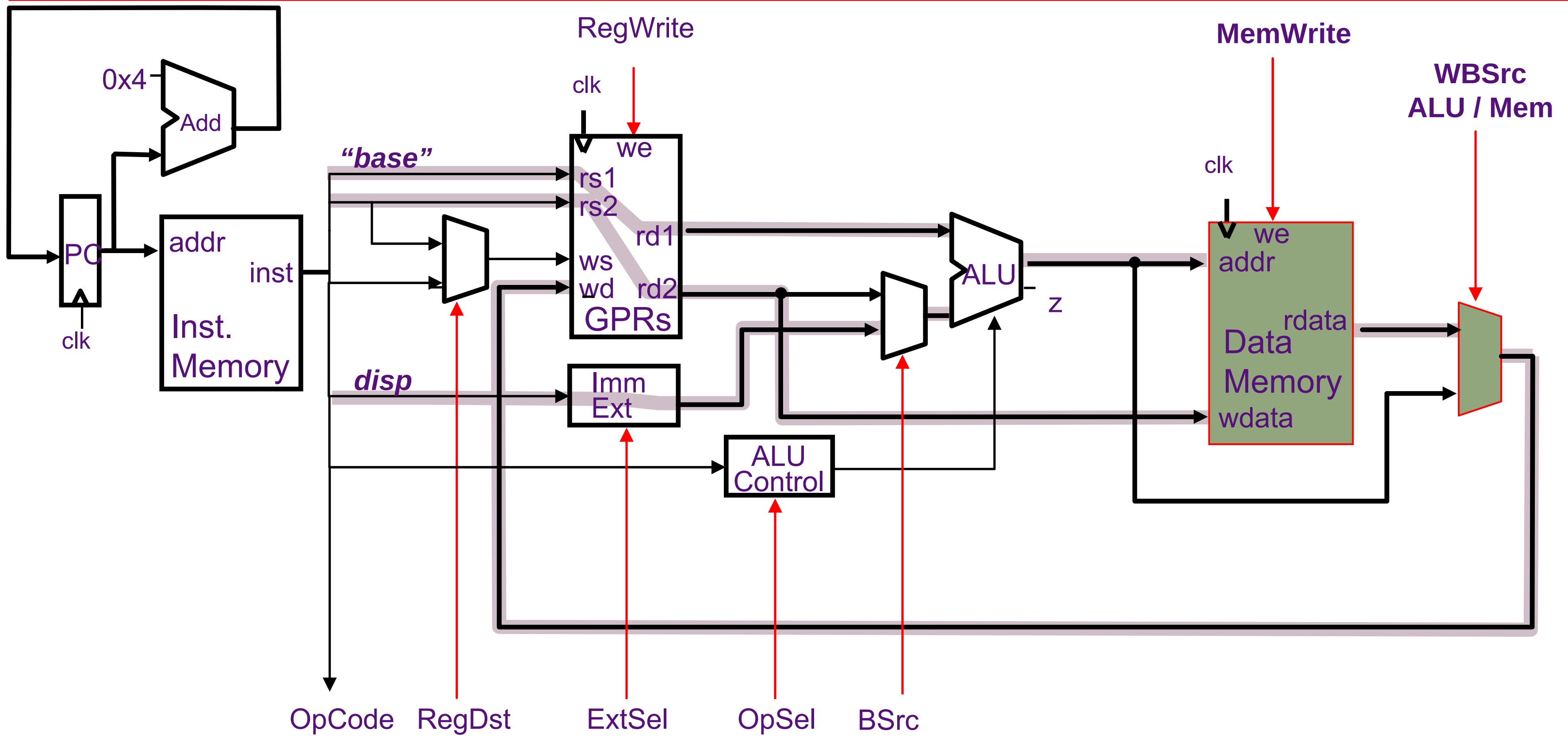
- read-only program memory
- read/write data memory

  at some level the two memories have to be the same

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

- A Load or Store instruction requires accessing the memory more than once during its execution

Load/Store Instructions:*Harvard Datapath*



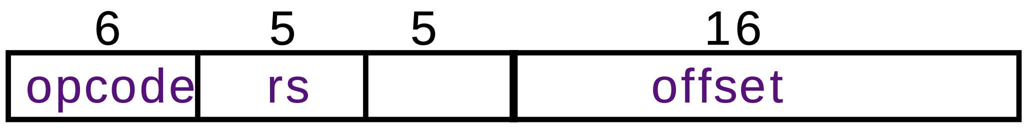
6	5	5	16
opcode	rs	rt	displacement
31 26	25 21	20 16	15 0

addressing mode
(rs) + displacement

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

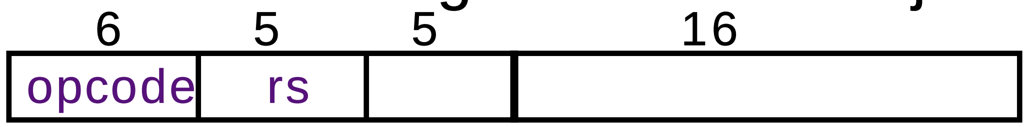
MIPS Control Instructions

Conditional (on GPR) PC-relative branch



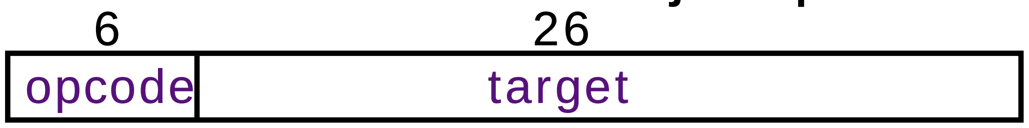
BEQZ, BNEZ

Unconditional register-indirect jumps



JR, JALR

Unconditional absolute jumps

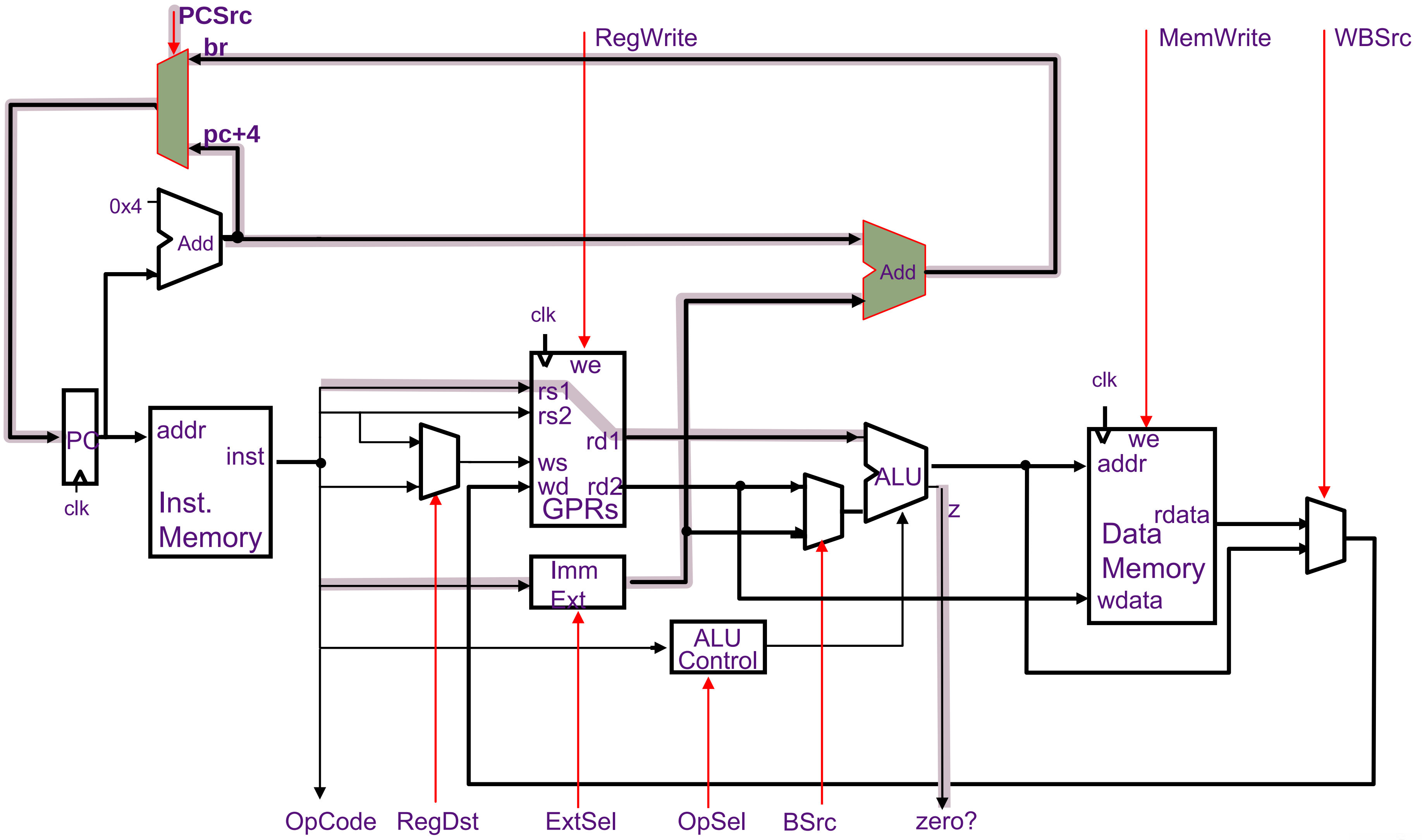


J, JAL

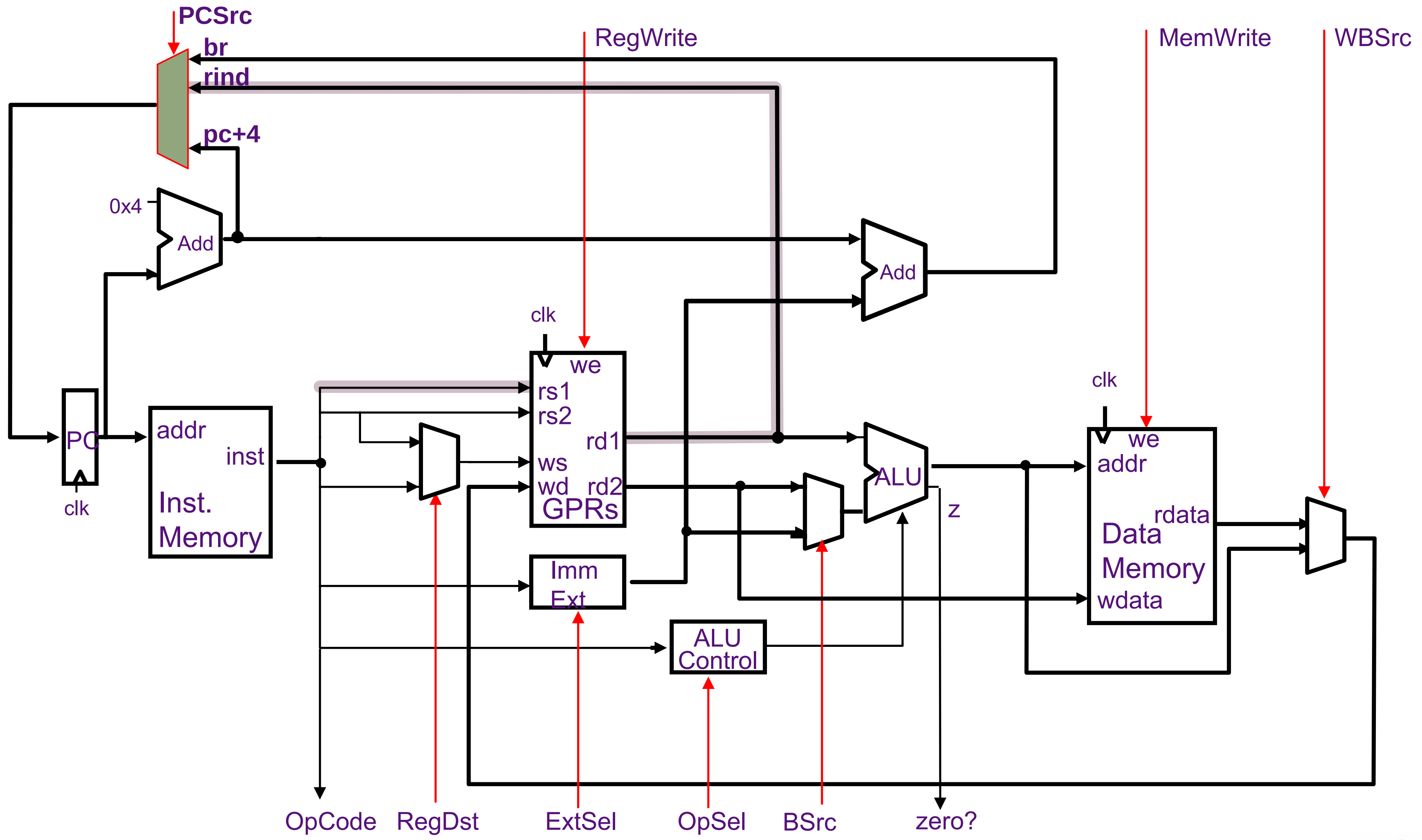
- PC-relative branches add offset$\times$4 to PC+4 to calculate the target address (offset is in words): $\pm$128 KB range
- Absolute jumps append target$\times$4 to PC<31:28> to calculate the target address: 256 MB range
- jump-&-link stores PC+4 into the link register (R31)
- All Control Transfers are delayed by 1 instruction

  we will worry about the branch delay slot later

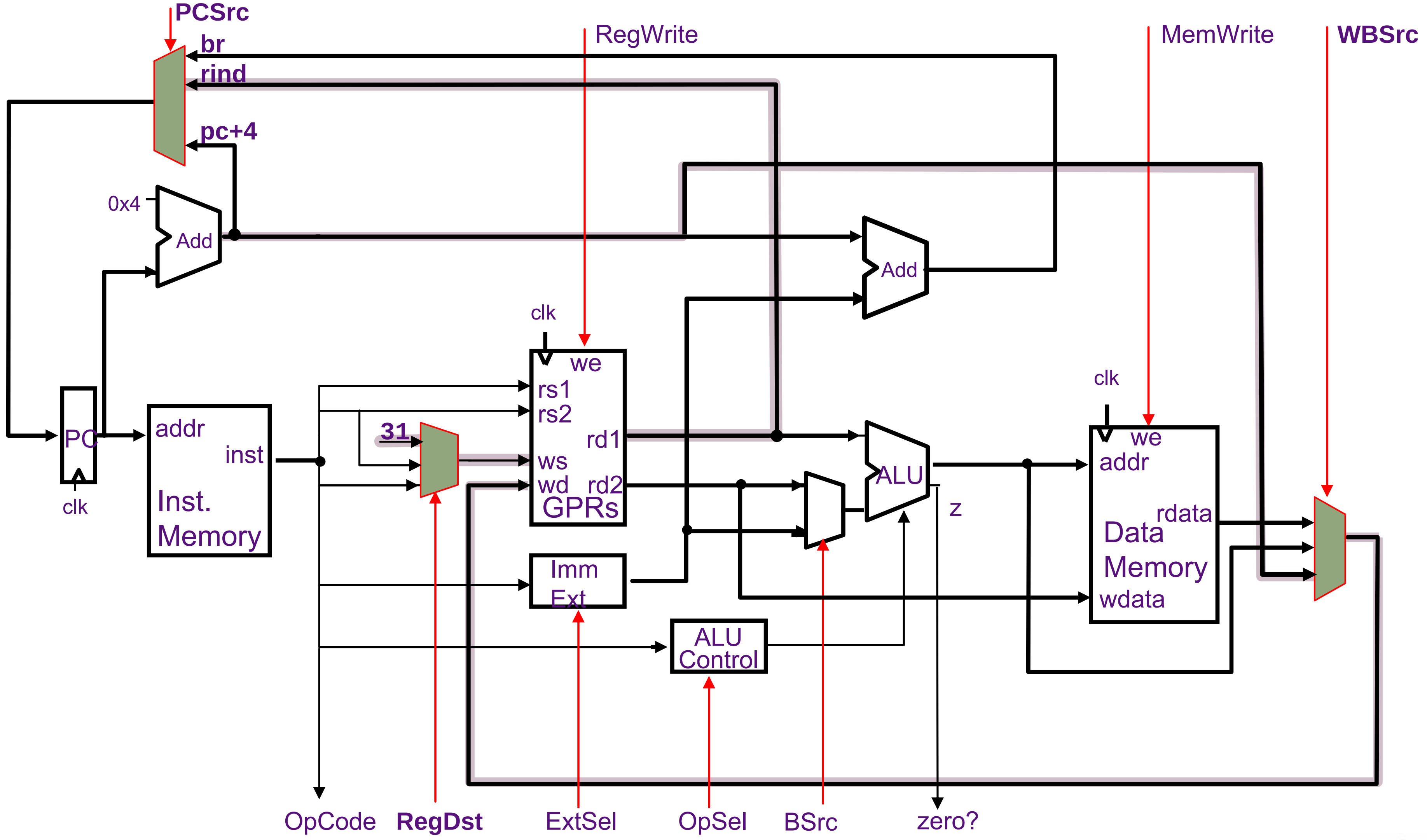
Conditional Branches (BEQZ, BNEZ)



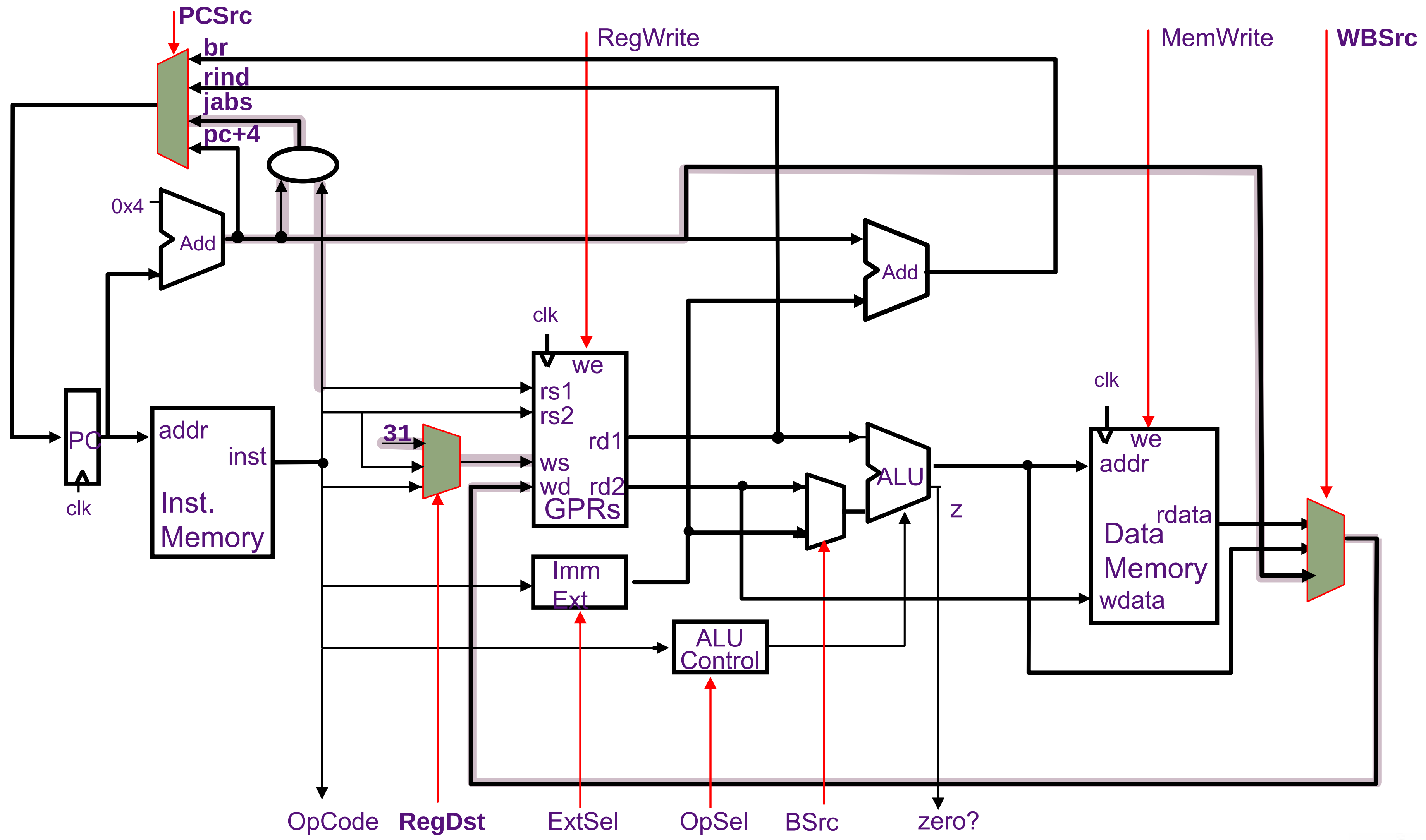
Register-Indirect Jumps (JR)



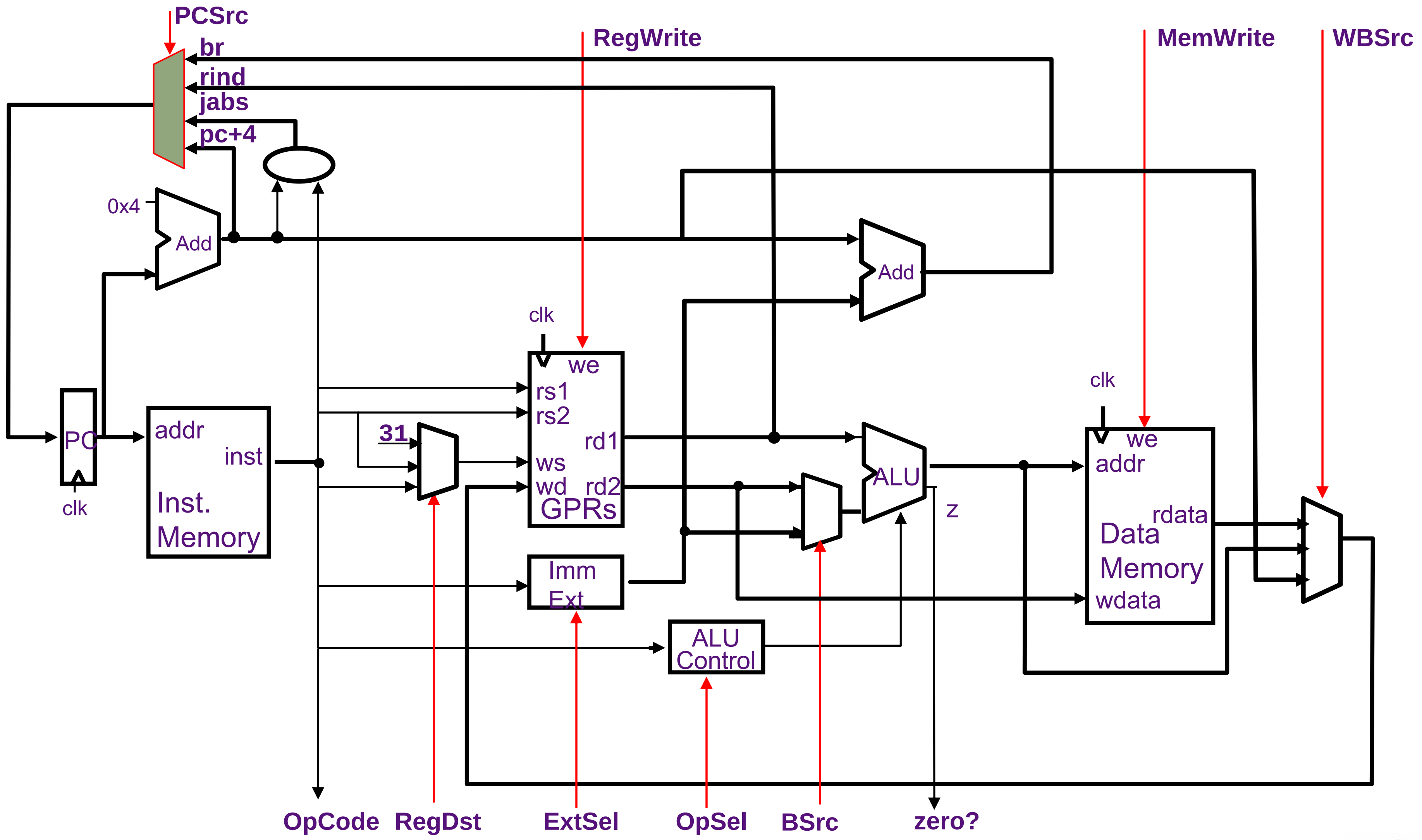
Register-Indirect Jump-&-Link (JALR)



Absolute Jumps (J, JAL)



Harvard-Style Datapath for MIPS



Five-minute break to stretch your legs

Single-Cycle Hardwired Control:
Harvard architecture

We will assume

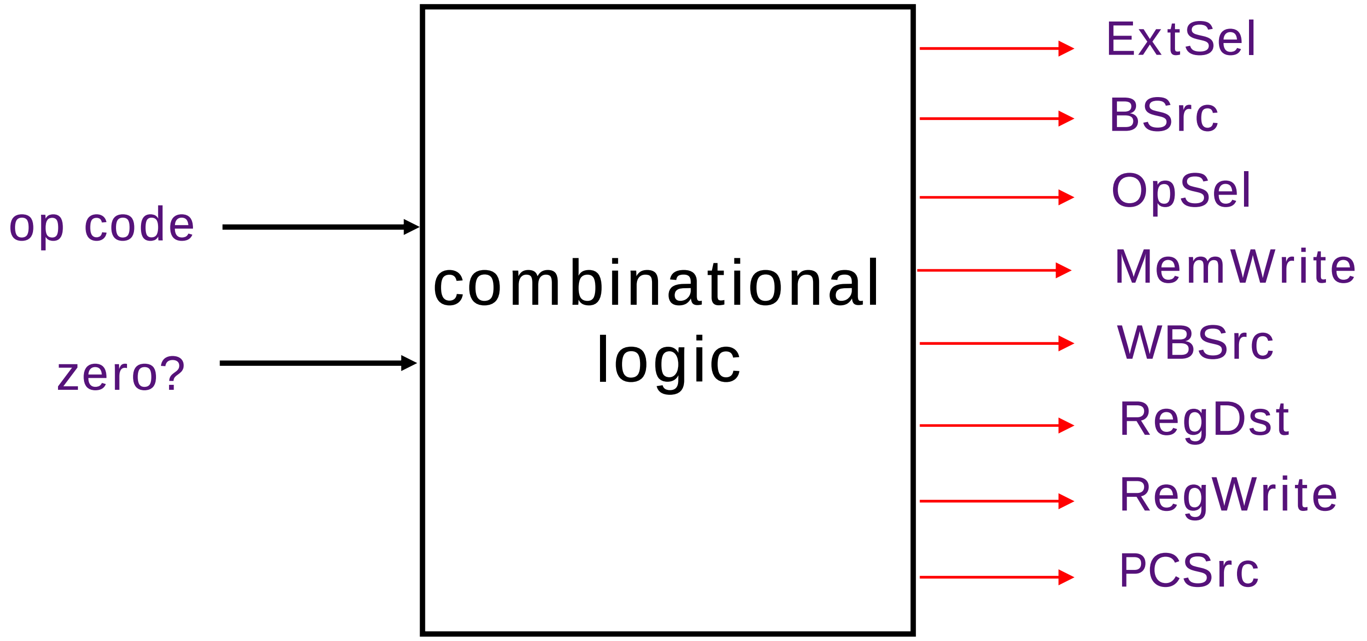
- clock period is sufficiently long for all of the following steps to be “completed”:

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

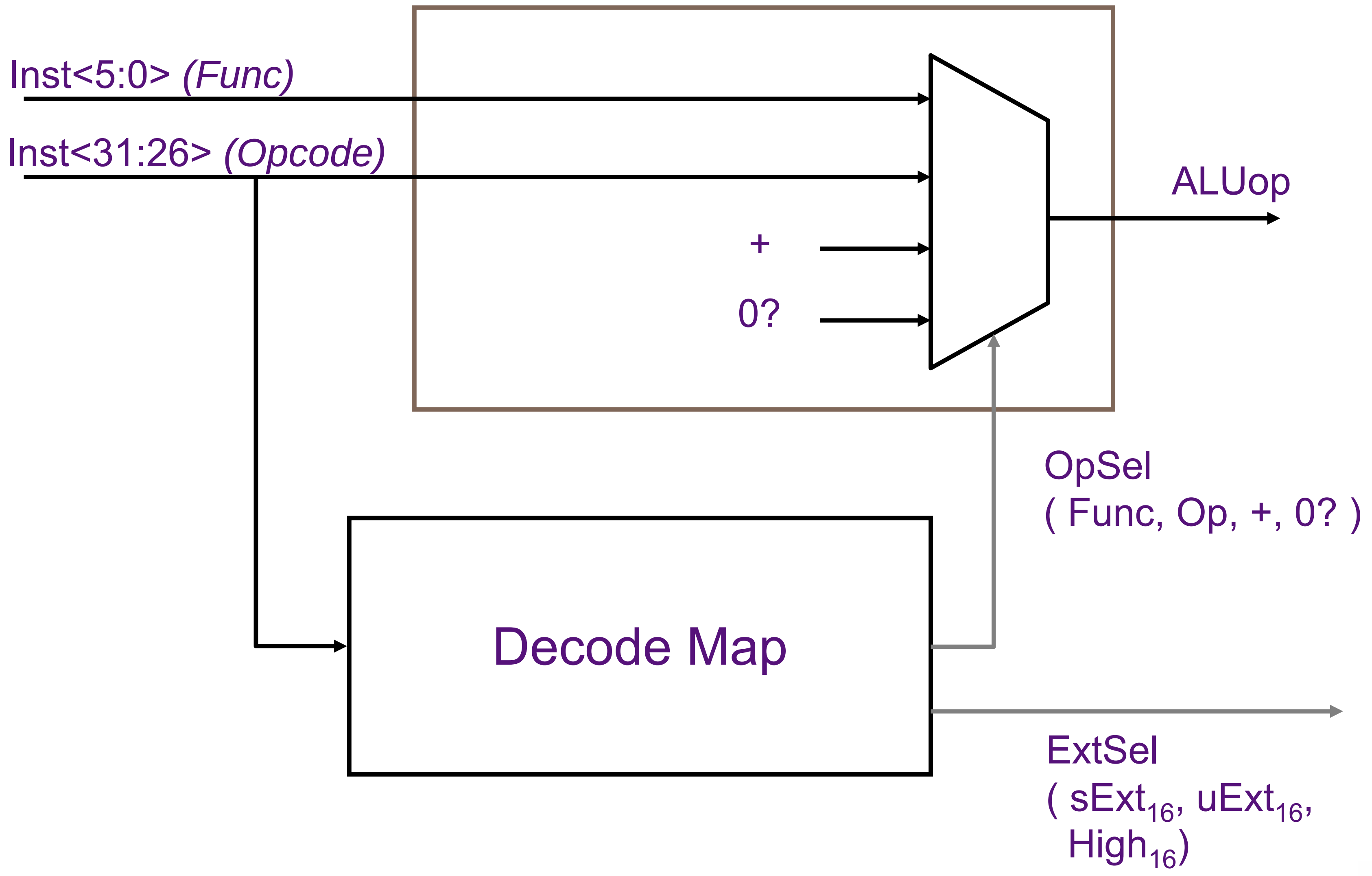
$$\Rightarrow \quad t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB}$$

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

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	$sExt_{16}$	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

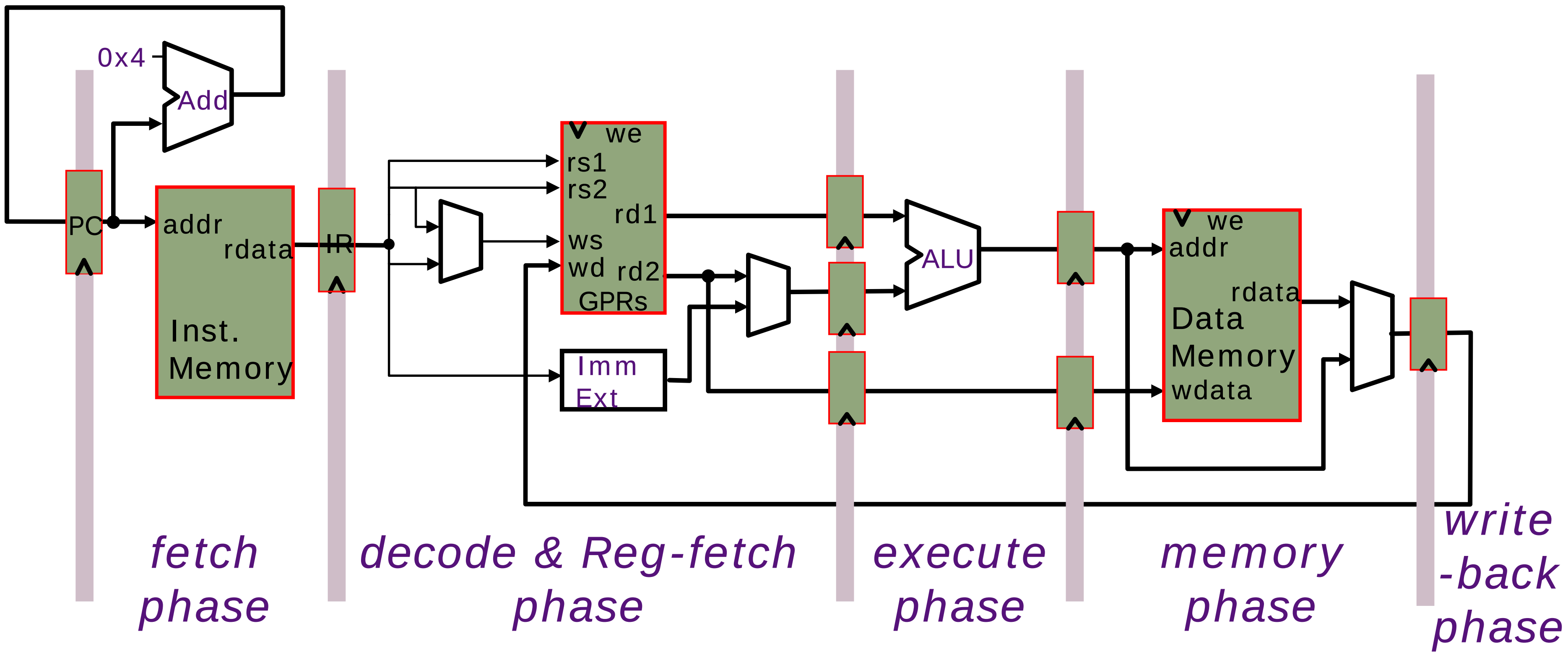
WBSrc = ALU / Mem / PC
PCSrc = pc+4 / br / rind / jabs

Pipelined MIPS

To pipeline MIPS:

- First build MIPS without pipelining with CPI=1
- Next, add pipeline registers to reduce cycle time while maintaining CPI=1

Pipelined Datapath

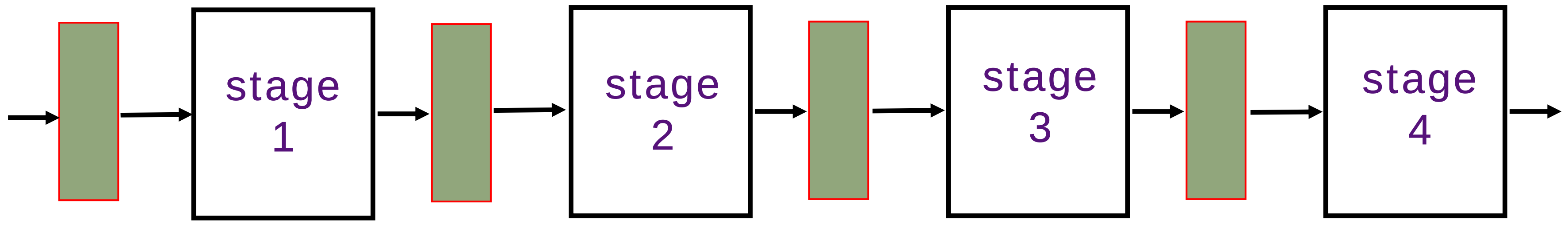


Clock period can be reduced by dividing the execution of an instruction into multiple cycles

$t_C > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\}$ ($= t_{DM}$ *probably*)

However, CPI will increase unless instructions are pipelined

An Ideal Pipeline



- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- The scheduling of an object entering the pipeline is not affected by the objects in other stages

These conditions generally hold for industrial assembly lines.
But can an instruction pipeline satisfy the last condition?

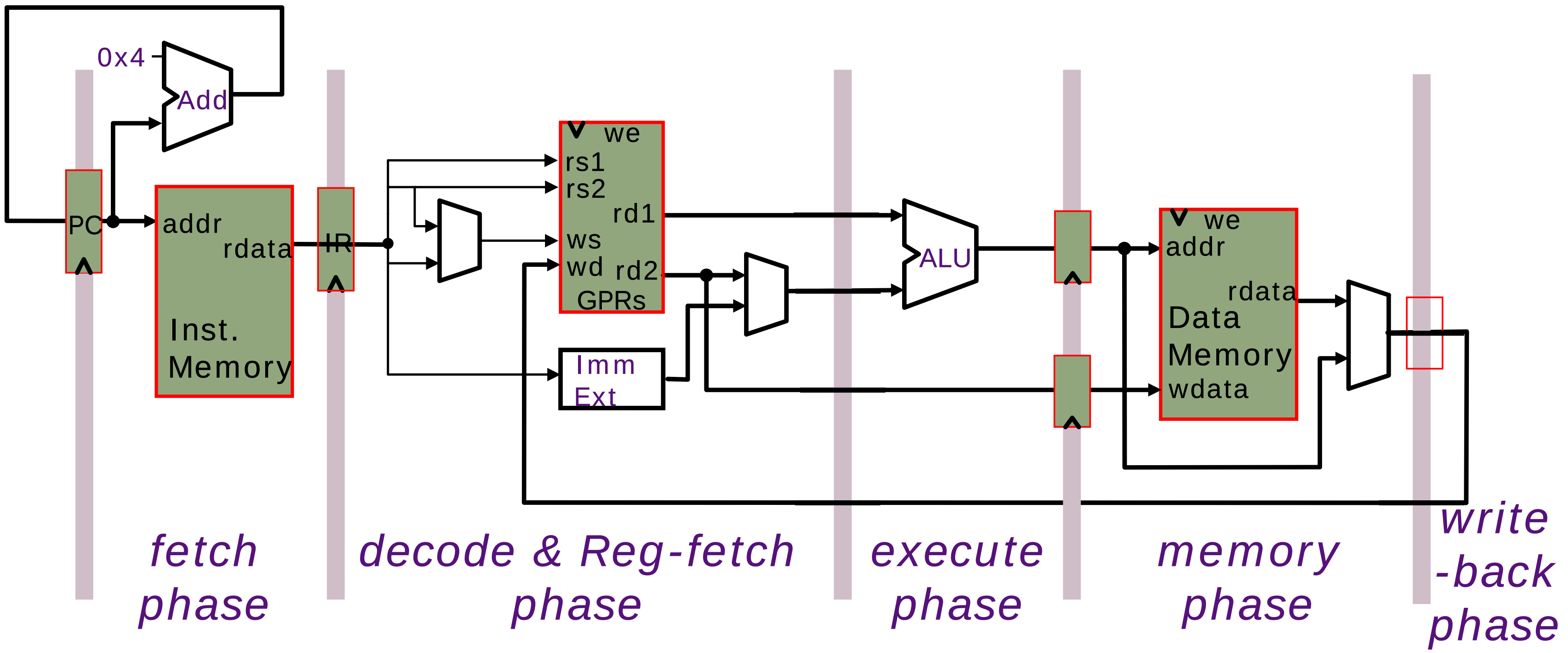
How to divide the datapath into stages

Suppose memory is significantly slower than other stages. In particular, suppose

t_{IM}	= 10 units
t_{DM}	= 10 units
t_{ALU}	= 5 units
t_{RF}	= 1 unit
t_{RW}	= 1 unit

Since the slowest stage determines the clock, it may be possible to combine some stages without any loss of performance

Alternative Pipelining



$t_C > \max\{t_{IM}, t_{RF}+t_{ALU}, t_{DM}+t_{RW}\} \quad = t_{DM}+t_{RW}$

$\Rightarrow$ *increase the critical path by 10%*

Write-back stage takes much less time than other stages. Suppose we combined it with the memory phase

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_C	Pipelined t_C	Speedup
1. $t_{IM} = t_{DM} = 10$, $t_{ALU} = 5$, $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline	25	5	5.0

It is possible to achieve higher speedup with more stages in the pipeline.

Thank you !