EECS 322 Computer Architecture

The Single Cycle Processor

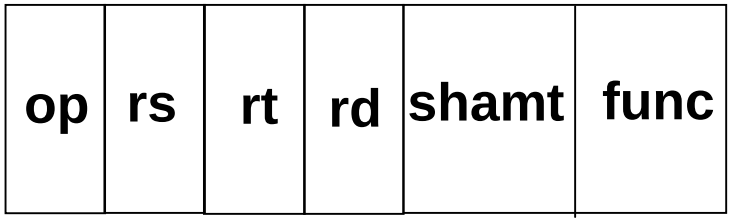
Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

MIPS fixed sized instruction formats

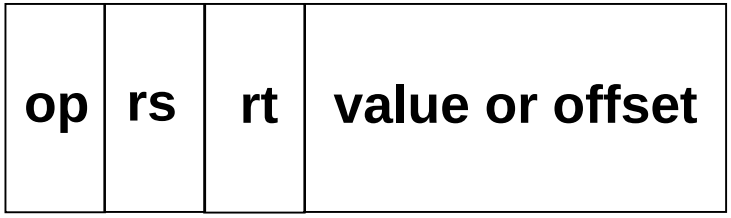
Appendix A & A.10

R - Format



ALU	alu $rd,$rs,$rt
jr	jr $rs

I - Format



ALUi	alui $rt,$rs,value
Data Transfer	lw $rt,offset($rs) sw $rt,offset($rs)
Branch	beq $rs,$rt,offset

J - Format

Jump	j address
Jump&Link	jal address

Review: MIPS instruction formats

Arithmetic

addi $rt, $rs, value

add $rd,$rs,$rt

jr $rs

Data Transfer

lw $rt,offset($rs)

sw $rt,offset($rs)

Conditional branch

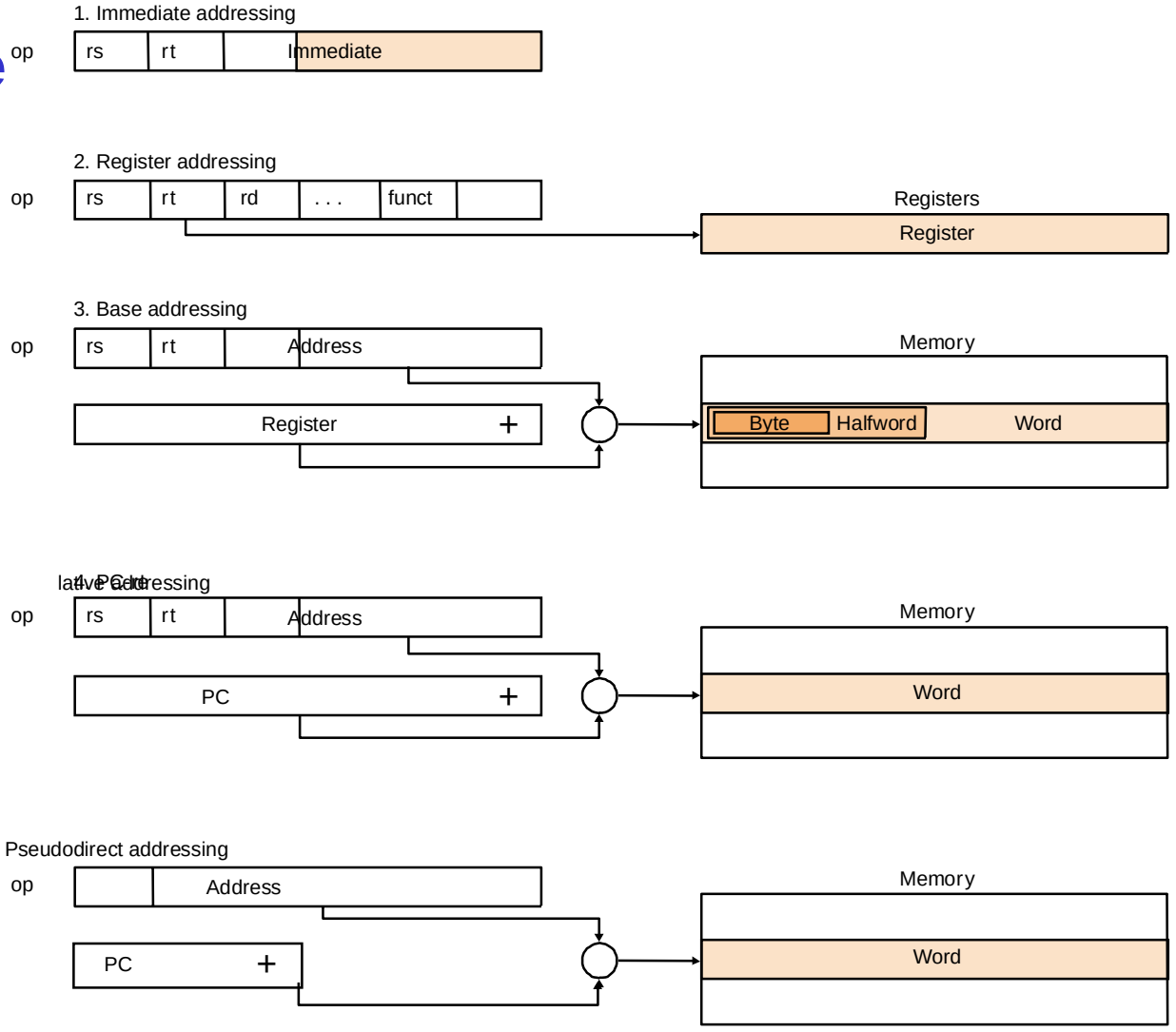
beq $rs,$rt,offset

bne $rs,$rt,offset

Unconditional jump

j address

jal address



MIPS instructions

δ^{32}=sign extend 16 bit number to 32 bits

ALU	alu $rd,$rs,$rt	$rd = $rs <alu> $rt
JR	jr $rs	$pc = $rs
ALUi	alui $rd,$rs,value16	$rd = $rs <alu> δ^{32}(value16)
Data Transfer	lw $rt,offset16($rs)	$rt = Mem[$rs + δ^{32}(offset16)]
	sw $rt,offset16($rs)	Mem[$rs + δ^{32}(offset16)]=$rt
Branch	beq $rs,$rt,offset16	$pc = ($rt == $rs)? ($pc+4+(δ^{32}(offset16)<<2))):($pc+4);
Jump	j address	$pc=($pc+4 & 0xF000000)|(addr<<2)
Jump&Link	jal address	$ra = $pc+4; $pc=($pc+4 & 0xF000000)|(addr<<2)

Assembling JUMP Instructions

Jump	j address	$pc= ($pc+4 & 0xF000000) | (addr<<2)

Suppose the fib_exit = 0x81fc084C, pc = 0x81fc08124,

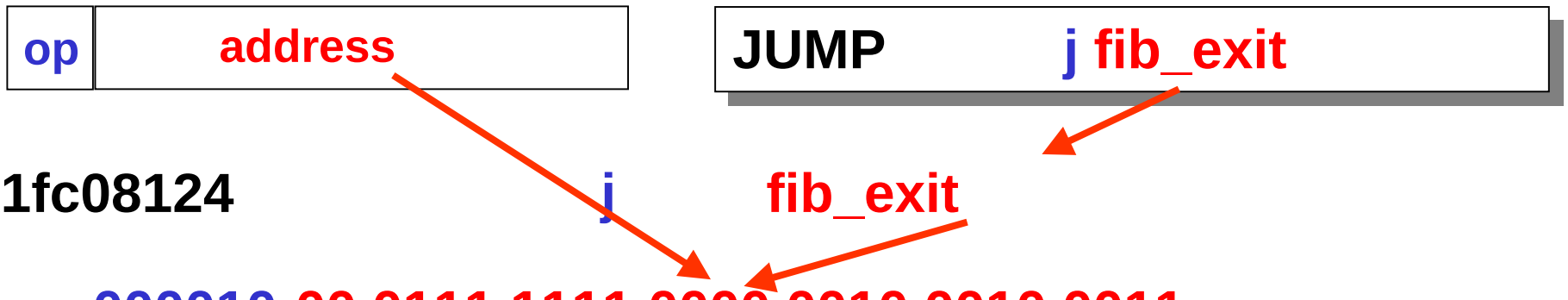
j fib_exit

Then addr>>2 = fib_exit >> 2 = 0x81fc084C >> 2

= 1000 0001 1111 1100 0000 1000 0100 1100>>2

= 0010 0000 0111 1111 0000 0010 0010 0011

= 0x27f0223 (upper 6 bits will be lost for opcode!)



Executing JUMP Instructions

Jump	j address	$pc= ($pc+4 & 0xF000000) | (addr<<2)

Suppose the pc = 0x81fc08124,

j 0x07f0223

000010:00 0111 1111 0000 0010 0010 0011

Then address =0x007f0223

Then address << 2 =0x01fc084C

Then $pc+4 = 0x81fc08128

Then $pc+4 & 0xF000000 = 0x800000000

Then $pc= ($pc+4 & 0xF000000) | (addr<<2) = 0x800000000 | 0x01fc084C

= 0x81fc084C

Executing JUMP Instructions different pc

Jump	j address	$pc= ($pc+4 & 0xF000000) | (addr<<2)

Same instruction but instead of pc = 0x81fc08124,

we have pc = 0x31fc08124

j 0x07f0223

Then address =0x007f0223

Then address << 2 =0x01fc084C

Then $pc+4 = 0x31fc08128

Then $pc+4 & 0xF000000 = 0x30000000

Then $pc= ($pc+4 & 0xF000000) | (addr<<2) = 0x30000000 | 0x01fc084C

= 0x31fc084C

Completely different address! The jump is memory block dependant

Book example

Jump	j address	$pc= ($pc+4 & 0xF000000) | (addr<<2)

Book example (decimal)

j 10000

Then jump address $=10000_{10}$

Then address << 2 (or divide by 4) $=2500_{10}$

assembly $<j,2500_{10}>$ or $<2:2500_{10}>$

Execution assume pc=0x01fc08124,

Then $pc+4 = 0x01fc0812C

Then $pc+4 & 0xF000000 = 0x000000000

Then $pc= ($pc+4 & 0xF000000) | (addr<<2) = 0x000000000 | 10000_{10}

$= 10000_{10}$

Review: MIPS registers and conventions

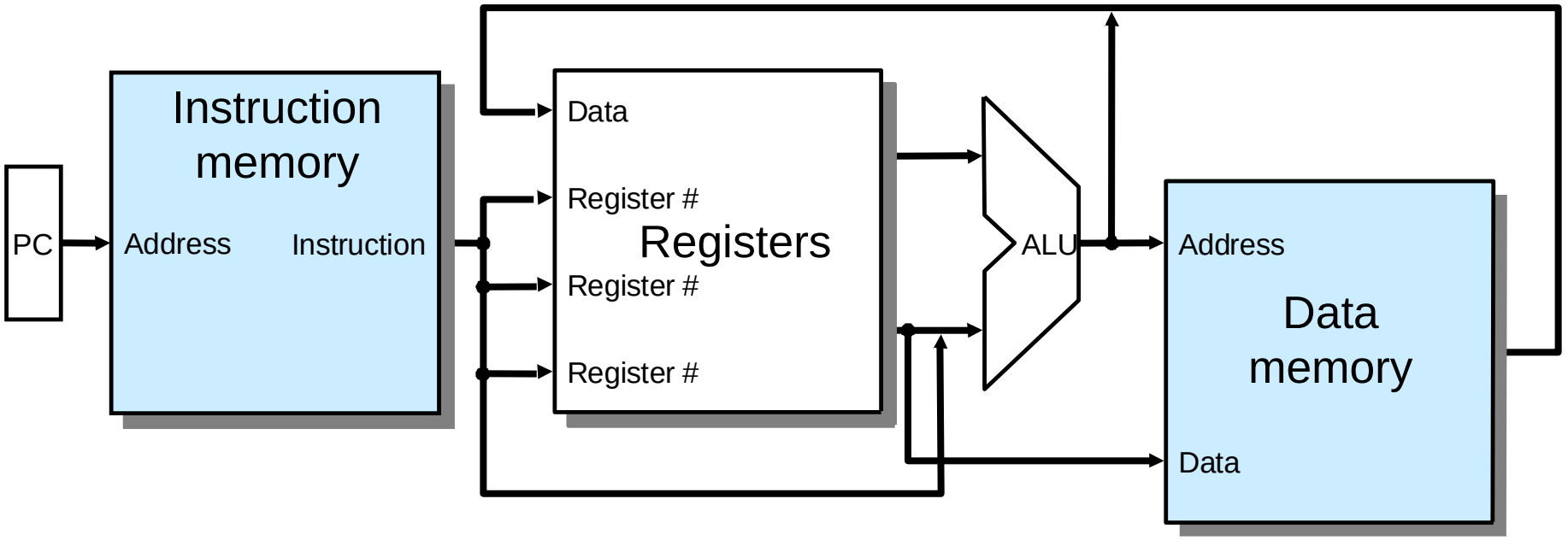
Name	Number	Conventional usage	SPARC
$0,$zero	0	Constant 0	%g0
$at	1	Temporary for pseudo-instructions	
$v0-$v1	2-3	Function results	%o0-%o1
$a0-$a3	4-7	Function arguments 1 to 4	%o0-%o6
$t1-$t9	8-15,24,35	Temporary	%g1-%g7
$s0-$s7	16-23	Saved Temporary	%L0-%L7
$k0-$k1	26-27	Reserved for OS kernel	
$gp	28	Pointer to global area	
$sp	29	Stack pointer	
$fp	30	Frame pointer	
$ra	31	Function Return address	%o7

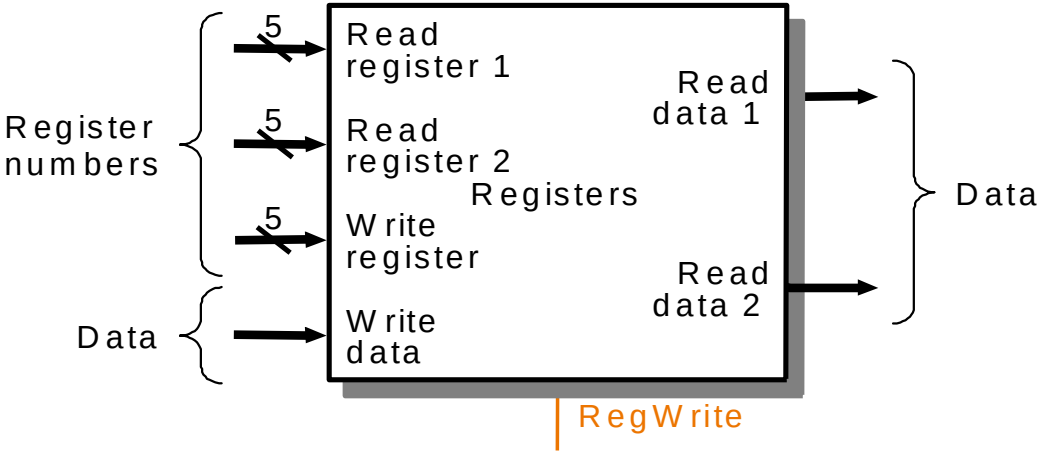
Review: Function calling

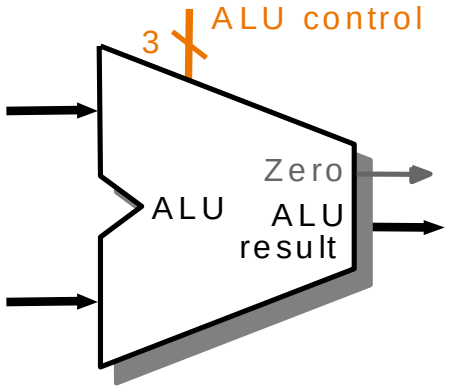
- **Calling conventions**
  - **$v0:$v1 = f($a0, a1, $a2, $a3, 0($sp), 4($sp),…)**

- **Caller Bookkeeping:**
  - **Arguments** **$a0-$a3, 0($sp), n-4($sp)**
  - **Return address** **$ra**
  - **Call function** **jal label** **# $ra=pc+4; pc=label**
- **Callee Bookkeeping:**
  - **Not restored** **$t0 - $t9, $a0-$a3, $at**
  - **Restore caller's** **$s0 - $s7, $sp, $fp**
  - **Return value** **$v0, $v1**
  - **Return** **jr $ra** **# pc = $ra**

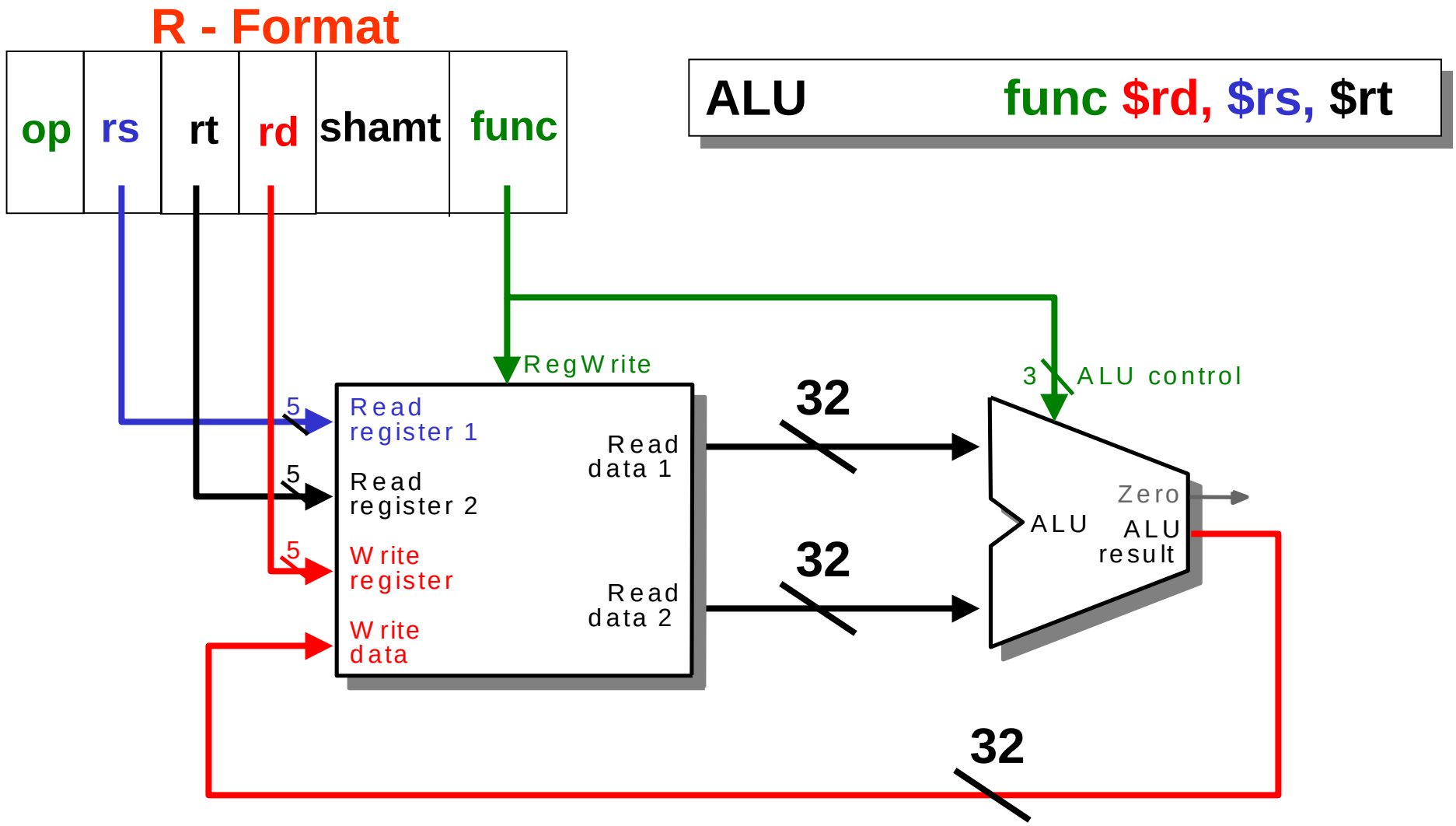
Abstract view of major functional units





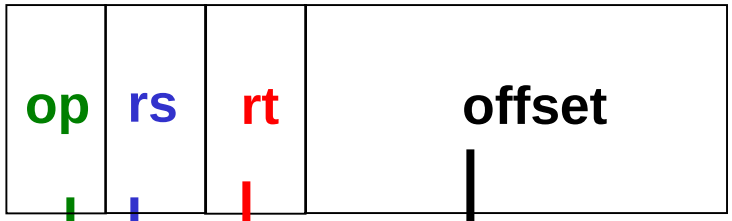


R-type instruction datapath

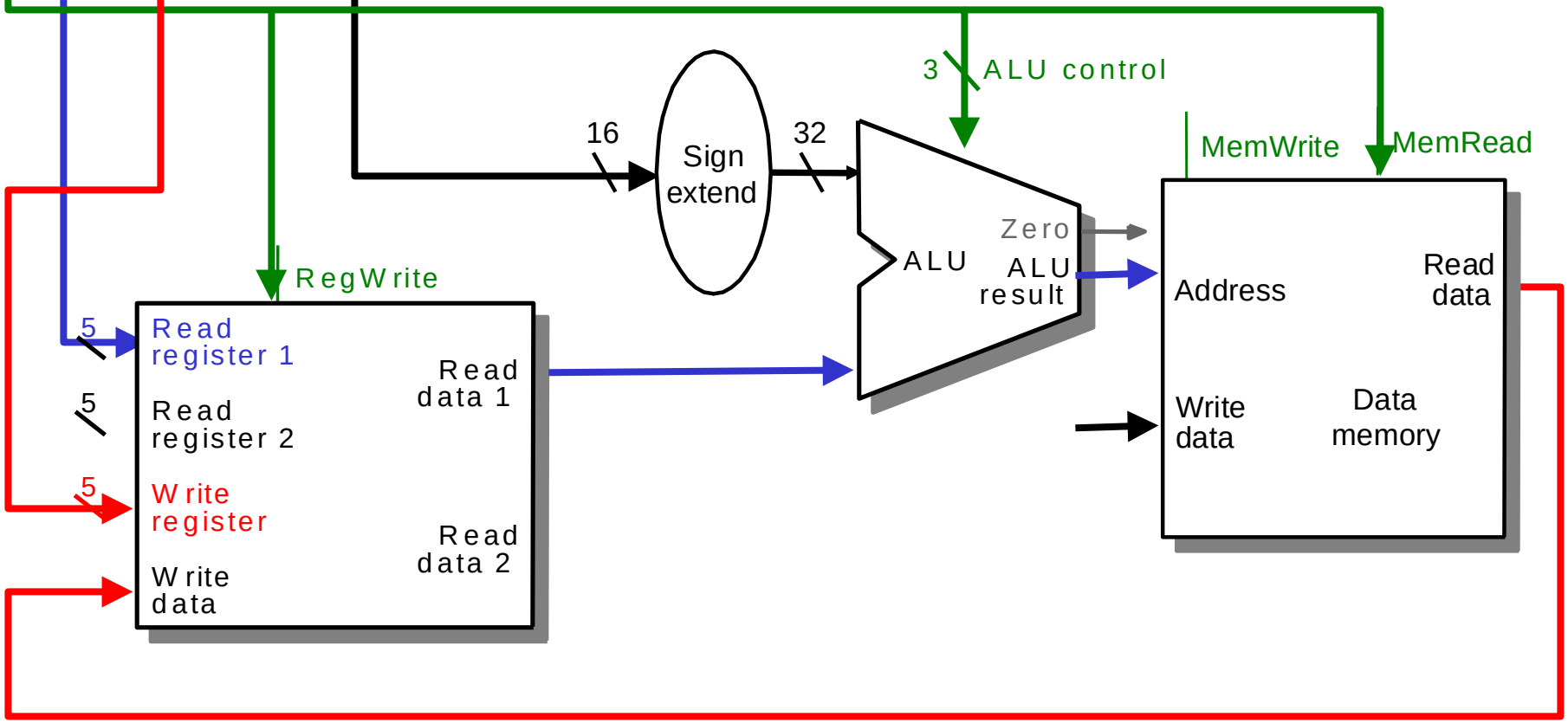


Lw I-type instruction datapath

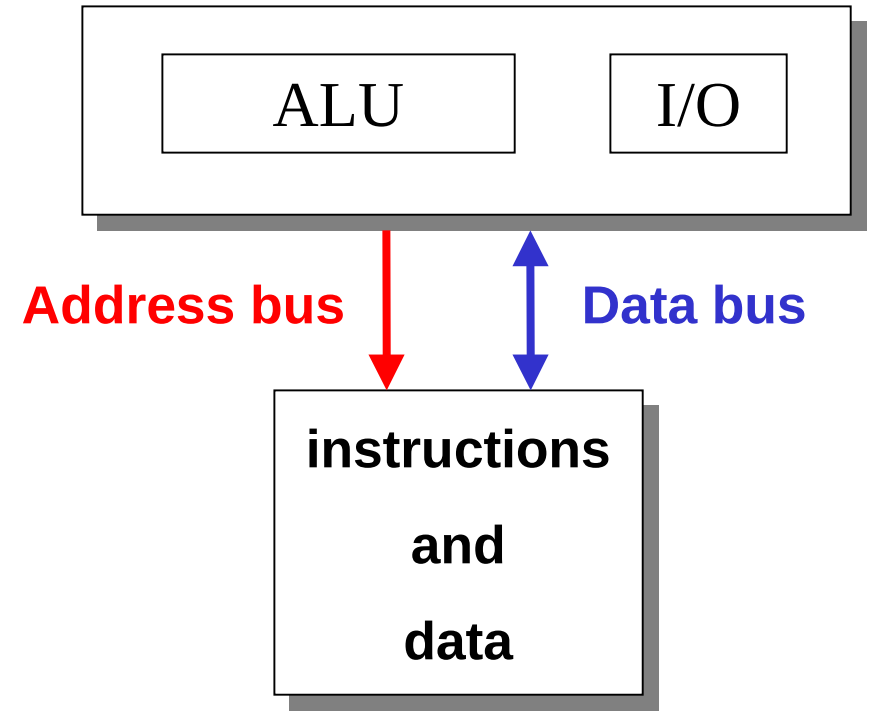
I - Format

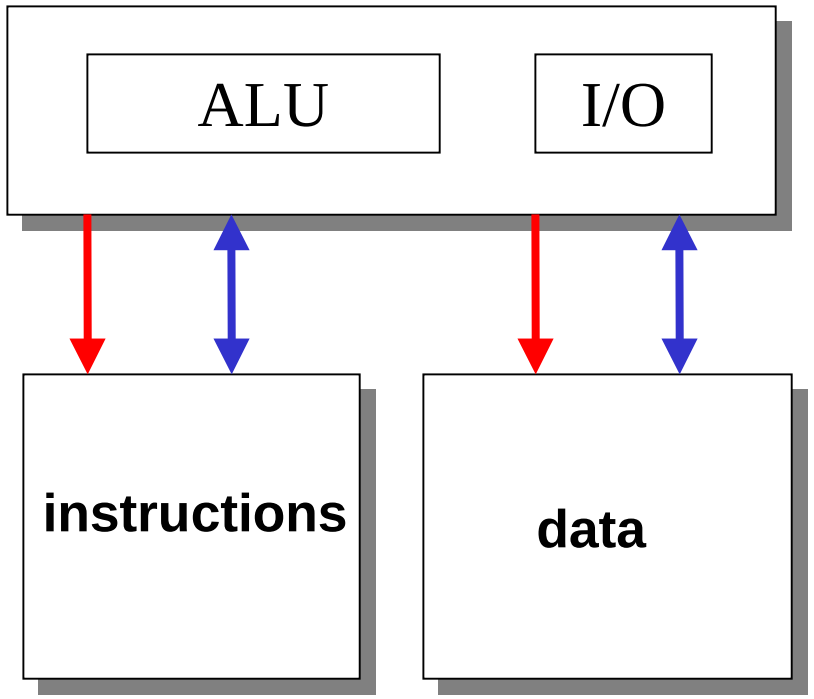


Data Transfer **lw $rt,offset($rs)**



Von Neuman & Harvard Architectures



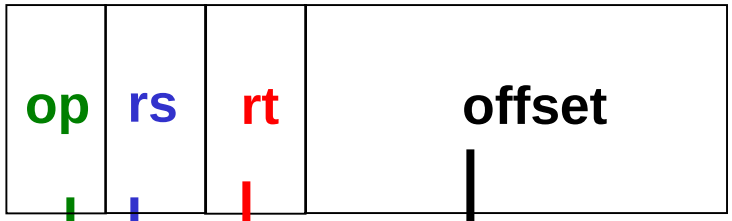


Von Neuman architecture **Area efficient but requires higher bus bandwidth because instructions and data must compete for memory.**

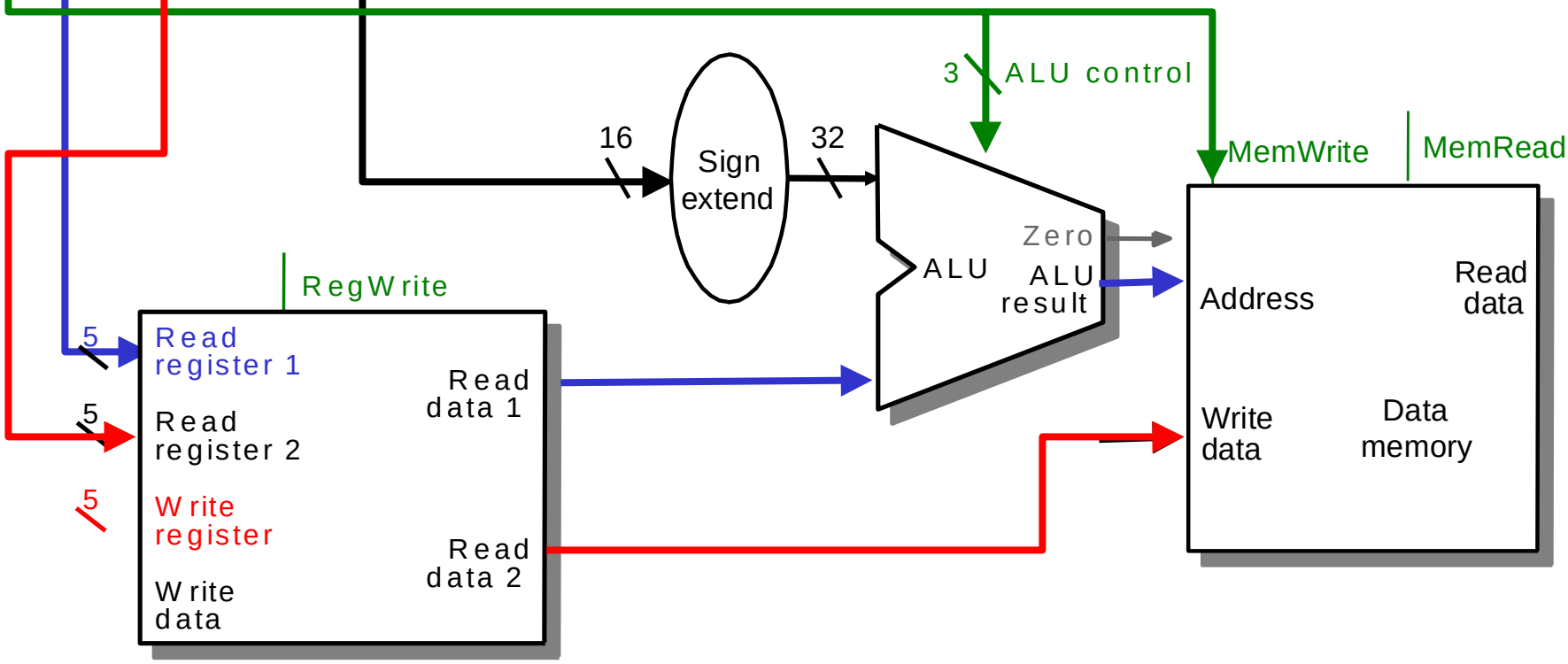
Harvard architecture **was coined to describe machines with separate memories. Speed efficient: Increased parallelism.**

Sw I-type instruction datapath

I - Format

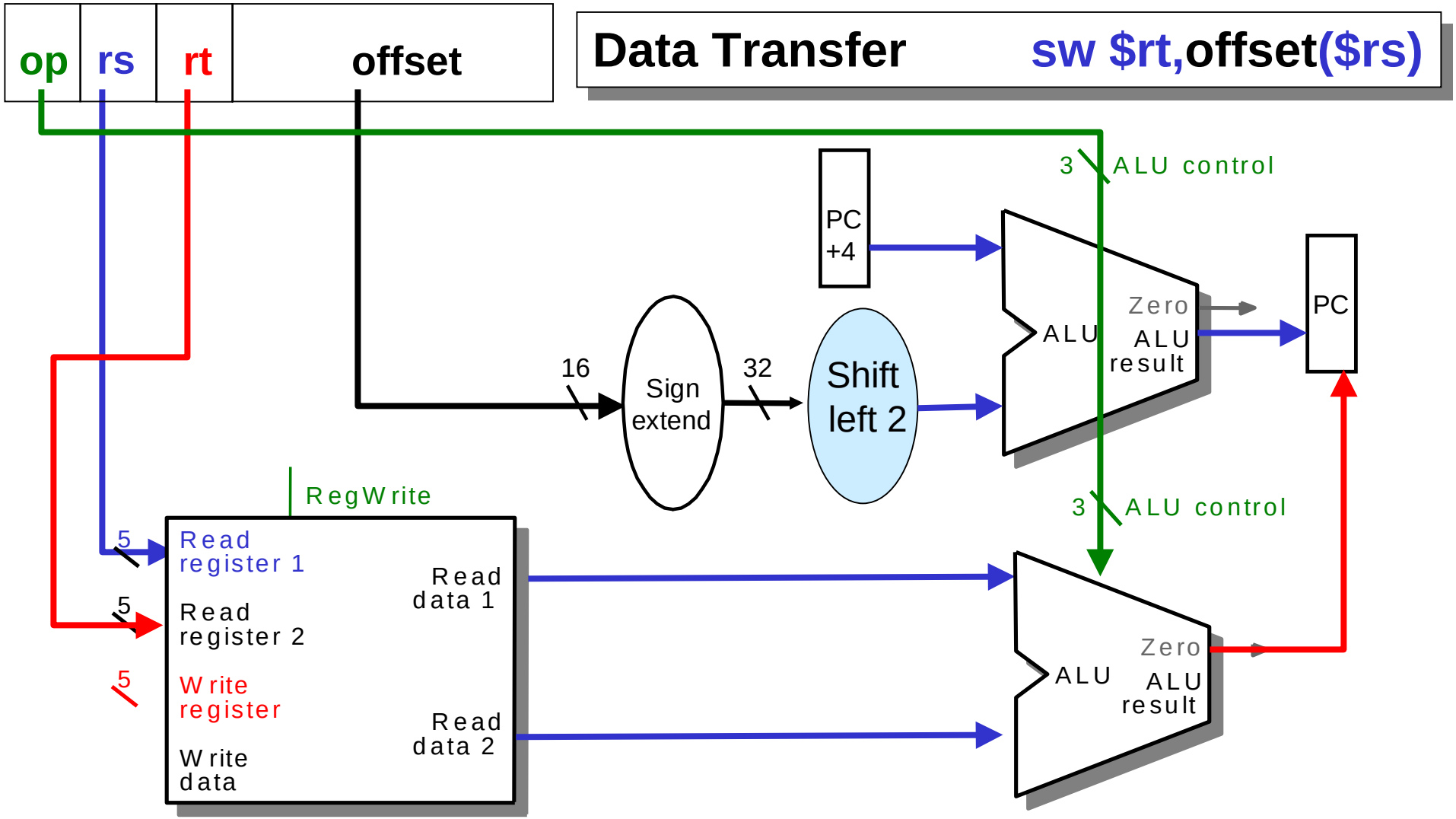


Data Transfer **sw $rt,offset($rs)**

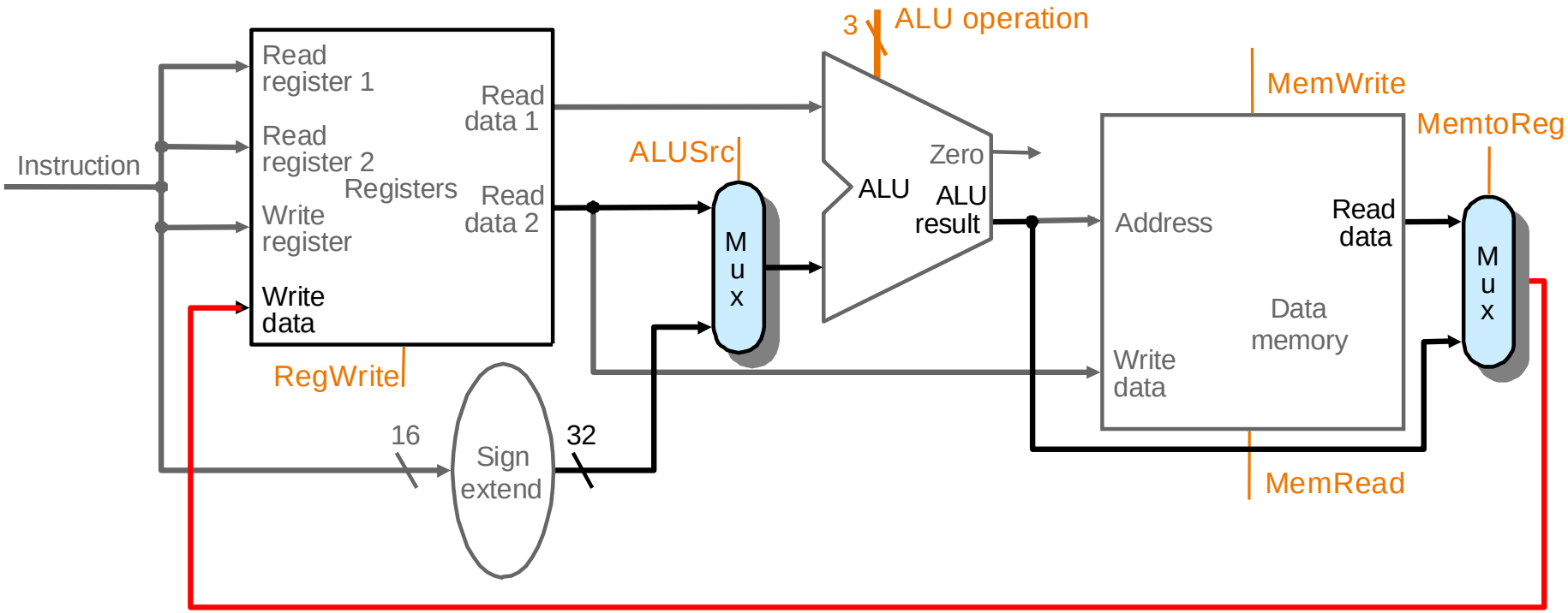


Branch I-type instruction datapath

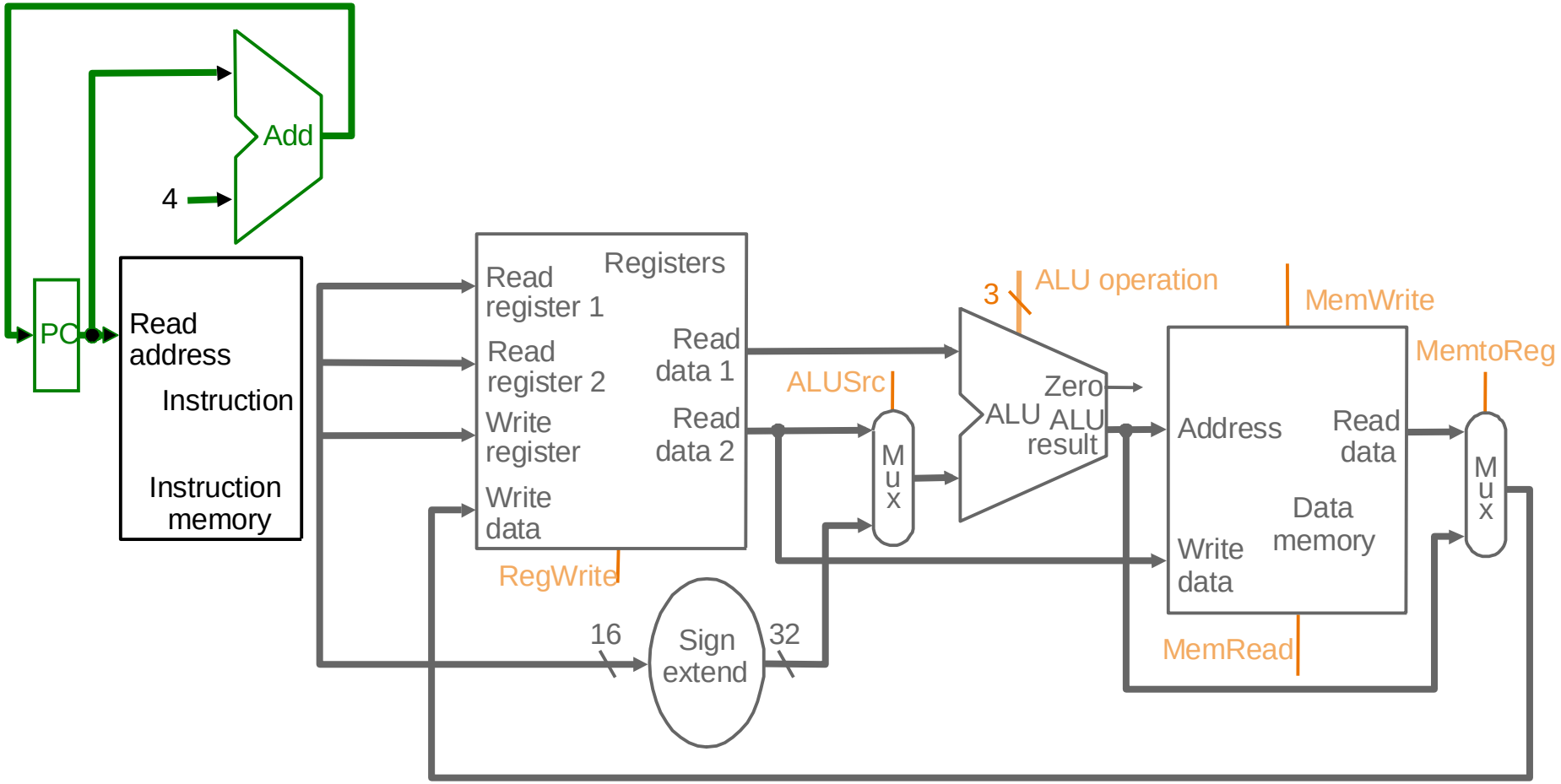
I - Format



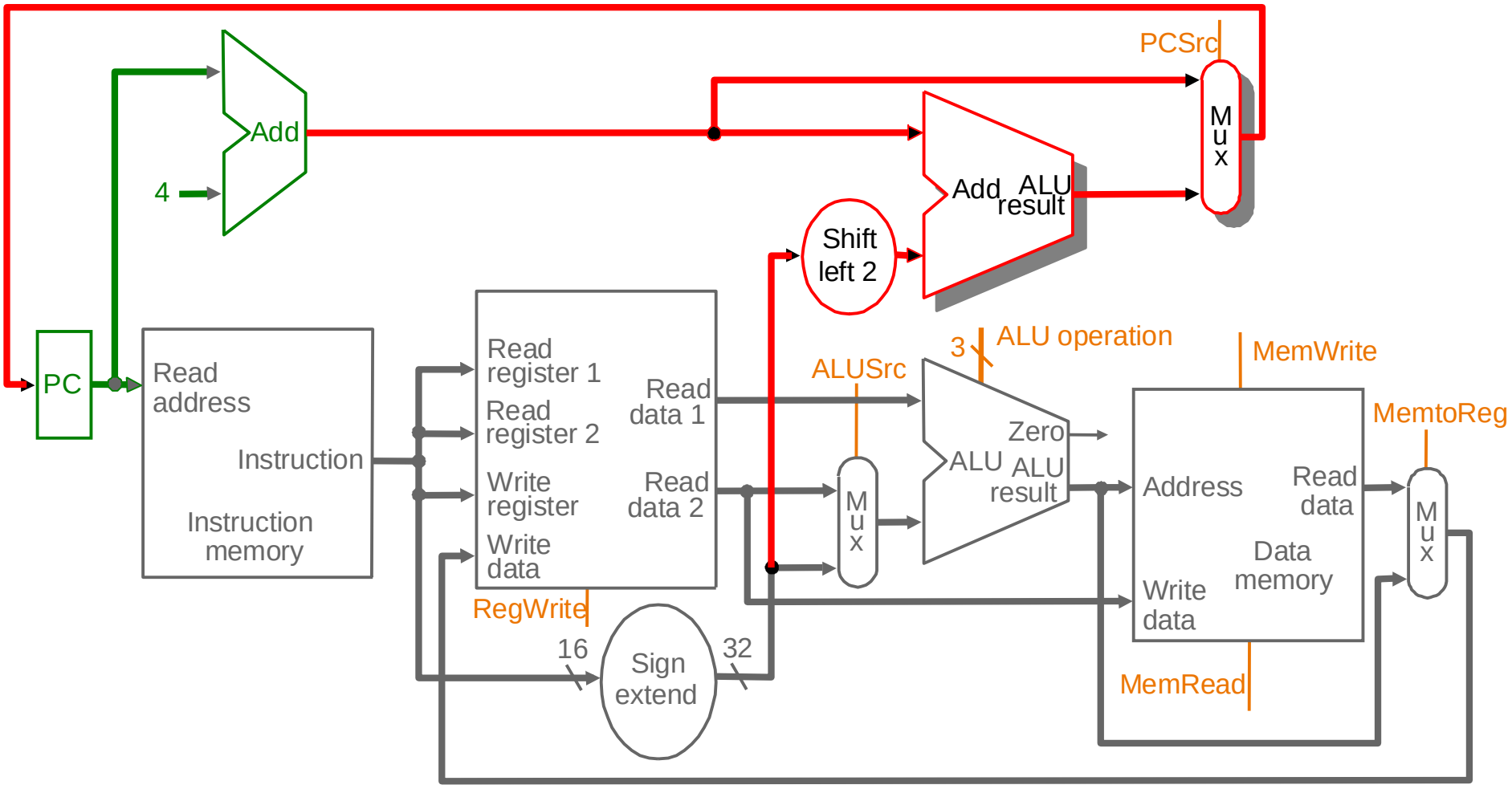
Muxiplexors: Combining datapaths



Instruction Fetch: pc=pc+4



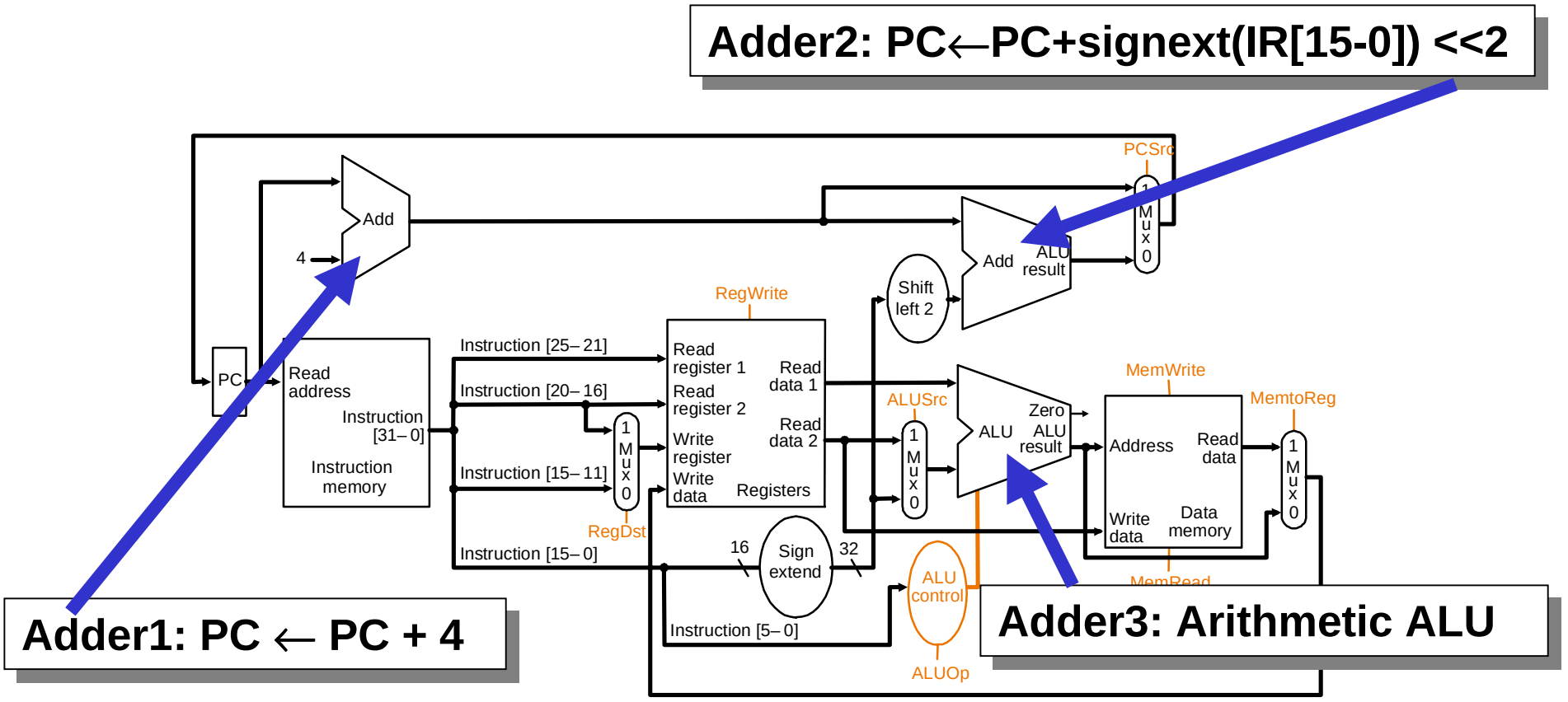
Combine Branch logic



Single Cycle Datapath

Single-cycle model (non-overlapping)

- Each instruction executes in a single cycle
- Every instruction and clock-cycle must be stretched to accommodate the slowest instruction (p.438)



Single Cycle = 2 adders + 1 ALU + 4 muxes

Reduced Instruction Set Computer

RISC - Reduced Instruction Set Computer

- By reducing the number of instructions that a processor supports and thereby reducing the complexity of the chip,
- it is possible to make individual instructions execute faster and achieve a net gain in performance
- even though more instructions might be required to accomplish a task.

RISC trades-off

instruction set complexity for instruction execution timing.

RISC Features

- **Large register set: having more registers allows memory access to be minimized.**
- **Load/Store architecture: operating data in memory directly is one of the most expensive in terms of clock cycle.**
- **Fixed length instruction encoding: This simplifies instruction fetching and decoding logic and allows easy implementation of pipelining.**

  All instructions are register-to-register format
  except Load/Store which access memory

  All instructions execute in a single cycle
  save branch instructions which require two.

  Almost all single instruction size & same format.

Complex Instruction Set Computer

CISC - Complex Instruction Set Computer

Philosophy: **Hardware is always faster than the software.**

Objective: **Instruction set should be as powerful as possible**

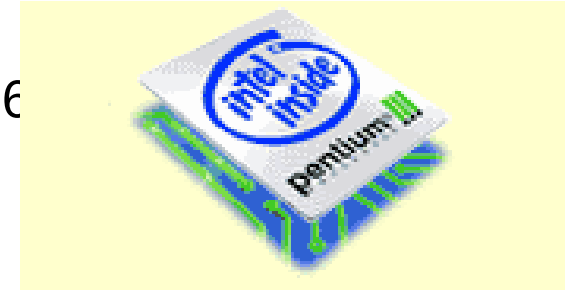
With a power instruction set, fewer instructions needed to complete (and less memory) the same task as RISC.

- **CISC was developed at a time (early 60's), when memory technology was not so advanced.**
- **Memory was small (in terms of kilobytes) and expensive.**

But for embedded systems, especially Internet Appliances, memory efficiency comes into play again, especially in chip area and power.

Reality Check: Intel 8086 clock cycles (1978)

Arithmetic			
3 clocks	add	reg16, reg16	
118-133 clocks	mul	dx:ax, reg16	very slow!!
128-154 clocks	imul	dx:ax, reg16	
114-162 clocks	div	dx:ax, reg16	
165-184 clocks	idiv	dx:ax, reg16	
Data Transfer			
14 clocks	mov	reg16, mem16	
15 clocks	mov	mem16, reg16	
Conditional Branch			
4/16 clocks	je	displacement8	
Unconditional Jump			
15 clocks	jmp	segment:offset16	



Comparison

CISC	RISC
Any instruction may reference memory	Only load/store references memory
Many instructions & addressing modes	Few instructions & addressing modes
Variable instruction formats	Fixed instruction formats
Single register set	Multiple register sets
Multi-clock cycle instructions	Single-clock cycle instructions
Less to no pipelining	Highly pipelined
Program code size small	Program code size large
	Memory Cache critical
Micro-program interprets instructions	Hardware (FSM) executes instructions
Complexity is in the micro-program	Complexity is in the compiler

Which is better...

RISC

Or

CISC

?

RISC versus CISC

RISC machines: SUN SPARC, SGI Mips, HP PA-RISC

CISC machines: Intel 80x86, Motorola 680x0

What really distinguishes RISC from CISC these days
lies in the architecture and not in the instruction set.

CISC occurs whenever there is a disparity in speed

- **between CPU operations and memory accesses**
- **due to technology or cost.**

What about combining both ideas?

Intel 8086 Pentium P6 architecture
is externally CISC but internally RISC & CISC!

Intel IA-64 executes many instructions in parallel.