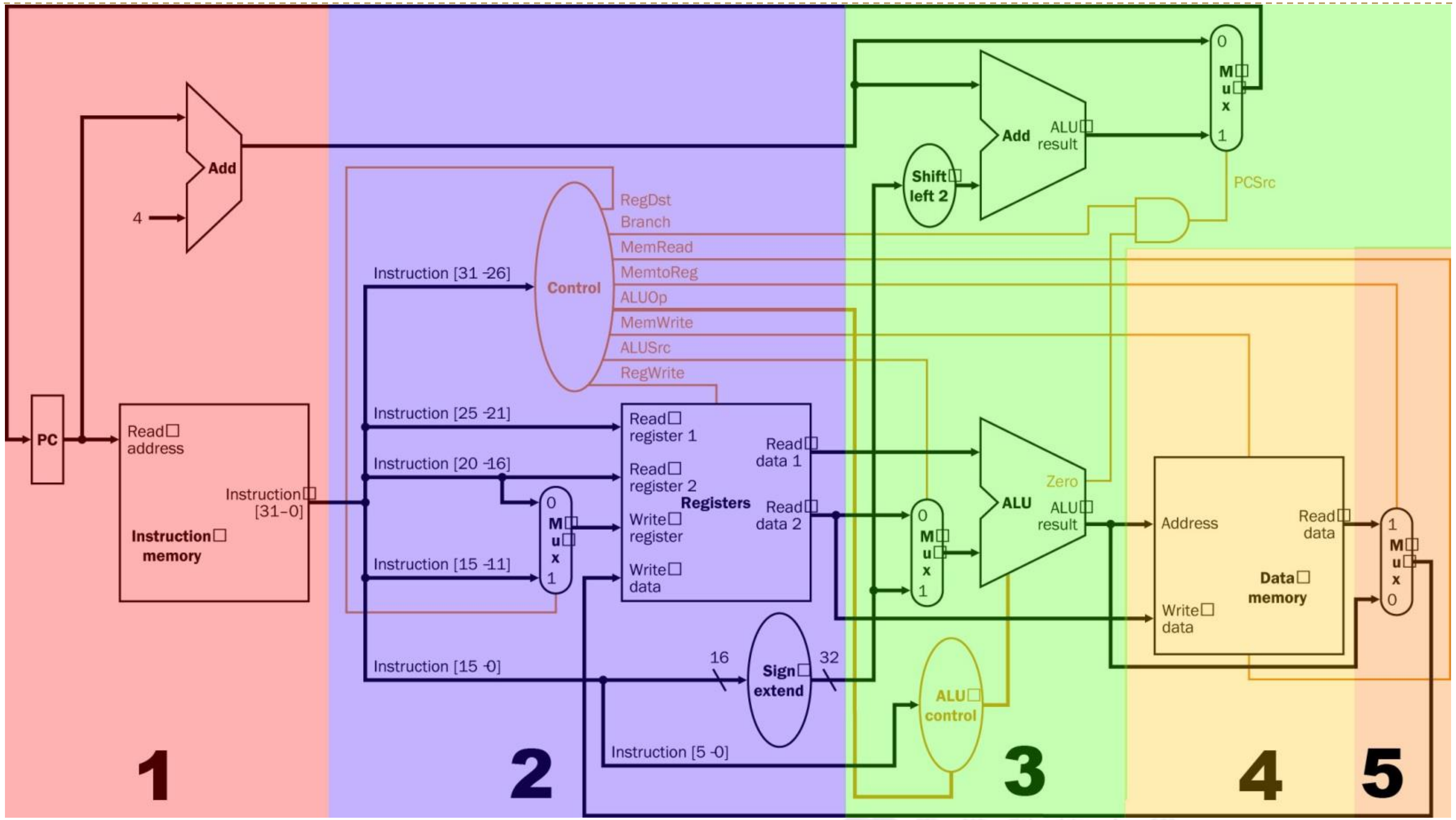
TUTORIAL 10 SINGLE-CYCLE DATAPATH AND CONTROL

Overview

- **We will review the following concept in this tutorial:**
- **Single-cycle implementation datapath**
  - Instruction fetch, decode and register read, execution, memory reference, write back
- **Single-cycle implementation control**
  - Instruction is executed through the guidance of the control signals
  - Effect of each control signal
- **Work with practical examples**
  - AND instruction
  - Exercises

5 Steps to execute an instruction



5 Steps to execute an instruction

- **Step 1: Instruction Fetch**
  - Fetch the instruction from the instruction memory pointed by the program counter (PC)
  - Do PC + 4 calculation
- **Step 2: Instruction Decode and Register Read**
  - Decode the instruction
  - Read source registers $rs and $rt
- **Step 1 and 2 are the same for all instructions**
- **Processor "knows what the instruction is" after the decoding step**

5 Steps to execute an instruction

- **Step 3: Execution**
  - R-type arithmetic/logic instructions: perform the required ALU operation
  - Memory-reference instructions: memory address calculation
  - Conditional branch instructions: comparison and branch calculation
- **Step 4: Memory Access (read/write)**
  - required only for the lw and the sw instructions
- **Step 5: Write Back**
  - Write back the result to destination register ($rt or $rd)
  - Update PC with PC+4 or branch target address (for conditional or un-conditional jump)

MIPS Green sheet

columns 3 and 4) together

MIPS Reference Data ①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

AND instruction Datapath

- Encode the instruction in 32-bit
- AND $t0, $t1, $t2

Encoding of this AND instruction (in decimal)

Opcode	RS	RT	RD	SHAMT	FUNC
0	9	10	8	0	36

Encoding of this AND instruction (in binary)

Opcode	RS	RT	RD	SHAMT	FUNC
000000	01001	01010	01000	00000	100100

AND instruction Datapath

- **AND instruction**

Description	Bitwise AND operation on the values of the registers \$rs and \$rt. Store the result in \$rd.
Operation	\$rd=\$rs & \$rt; PC=PC+4
Syntax	AND \$rd, \$rs, \$rt
Encoding	000000 01001 01010 01000 00000 100100 opcode rs rt rd shamt func

- **Example: AND \$t0, \$t1, \$t2**
  - Encoding 000000 01001 01010 01000 00000 100100

Step 1: Instruction Fetch

A = 32-bit, address of the instruction

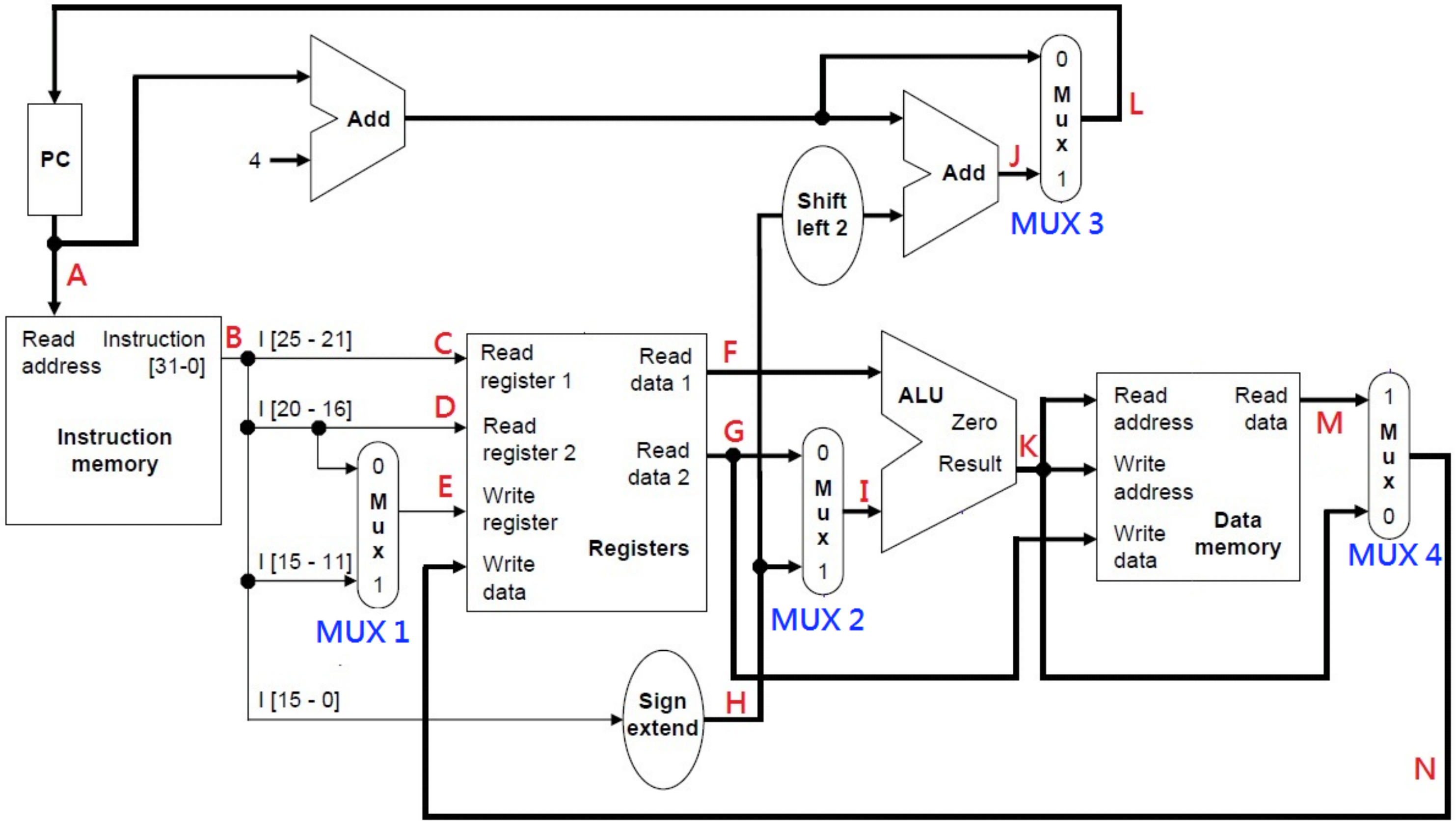
B= 32-bit, the fetched instruction

No control signals needed

Instruction fetched

000000 01001 01010 01000 00000 100100

opcode rs rt rd shamt func



Step 2: Instruction Decode and Register Read

C = 01001, D = 01010, E = 01000

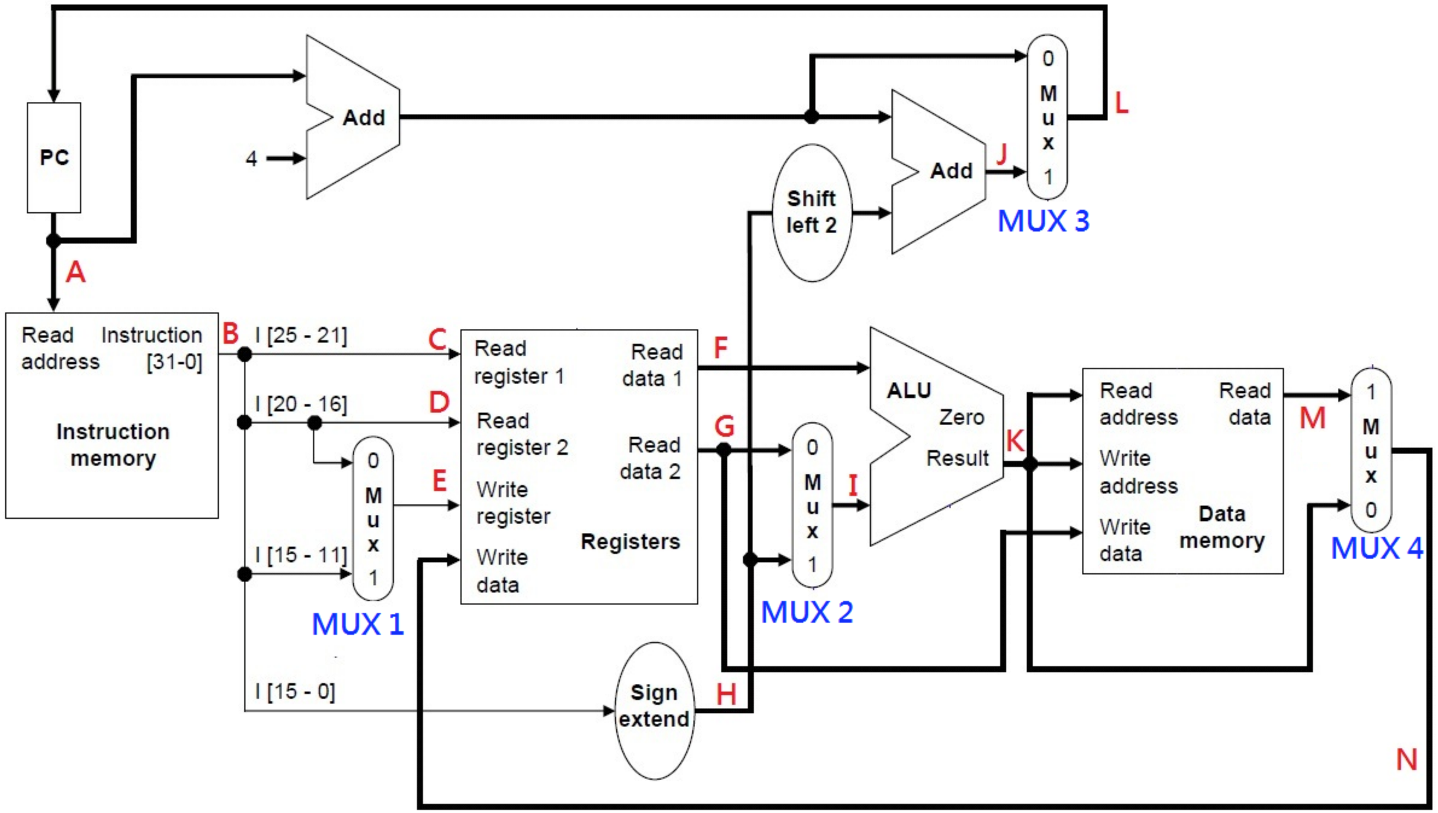
F = 32-bit, value of $rs

G = 32-bit value $rt

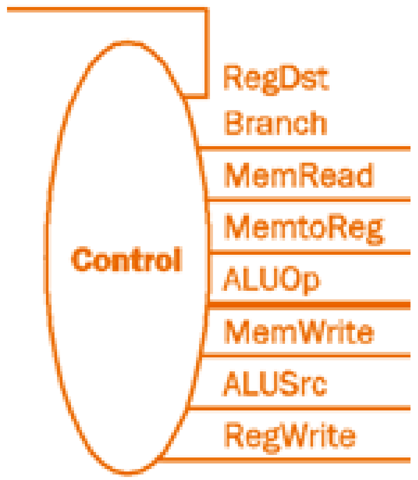
H= sign-extended lower half, 0000 0000 0000 0000 01000 00000 100100

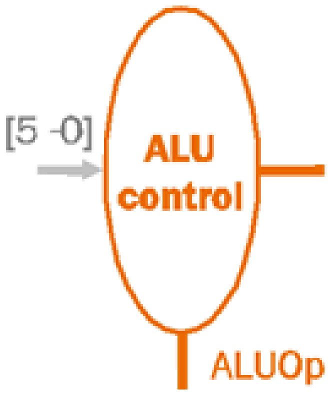
No control signals needed

Instruction fetched
000000 01001 01010 01000 00000 100100
opcode rs rt rd shamt func



Control Signals generated in ID Stage





Control signal	Value
RegDst	1 write back to $rd
Branch	0 sequential execution
MemRead	0 no memory access
MemtoReg	0 write back ALU output
ALUOp	10 (refer to func code to decide ALU operation)
MemWrite	0 no memory access
ALUSrc	0 2nd source operand is $rt
RegWrite	1 enable register write

Inputs		Outputs
Funct (bits 5:0)	ALUOp	ALU Control
100 100	10	0000 (AND operation)

Implementing ALU Control Block

- Assume 2-bit ALUOp derived from opcode
- Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

input input output

Step 3: Execution

I= G = 32-bit, value of $rt

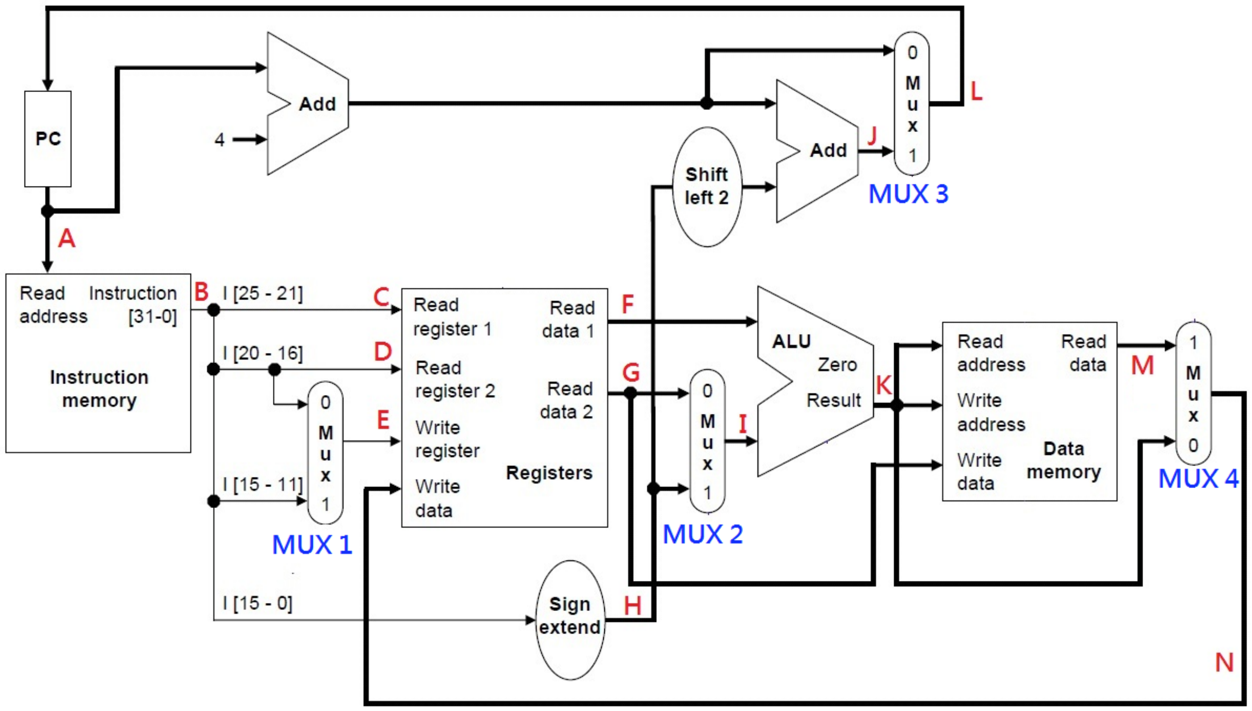
K = 32-bit, (value of $rs) bitwise_AND (value of $rt)

Instruction fetched
000000 01001 01010 01000 00000 100100
opcode rs rt rd shamt func

J = 32-bit, 0000 0000 0000 0001 0000 0000 1001 0000 + PC + 4

ALUSrc (MUX 2) = 0

ALU Control = 0000



Step 4: Memory Reference

No memory operation

memread = 0

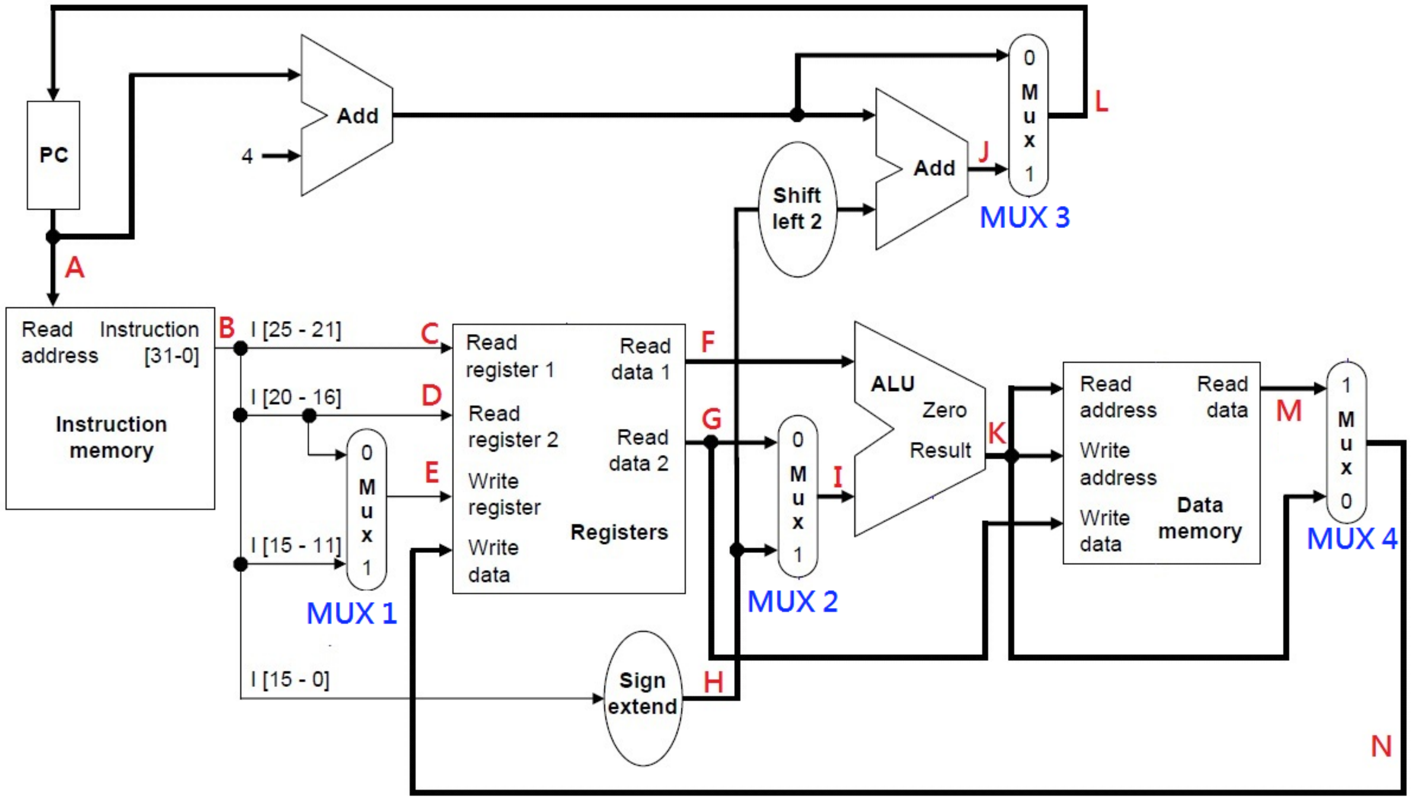
memwrite = 0

memtoreg (MUX 4) = 0

Instruction fetched

000000 01001 01010 01000 00000 100100

opcode rs rt rd shamt func



Step 5: Write back

N = K = 32-bit, ALU output

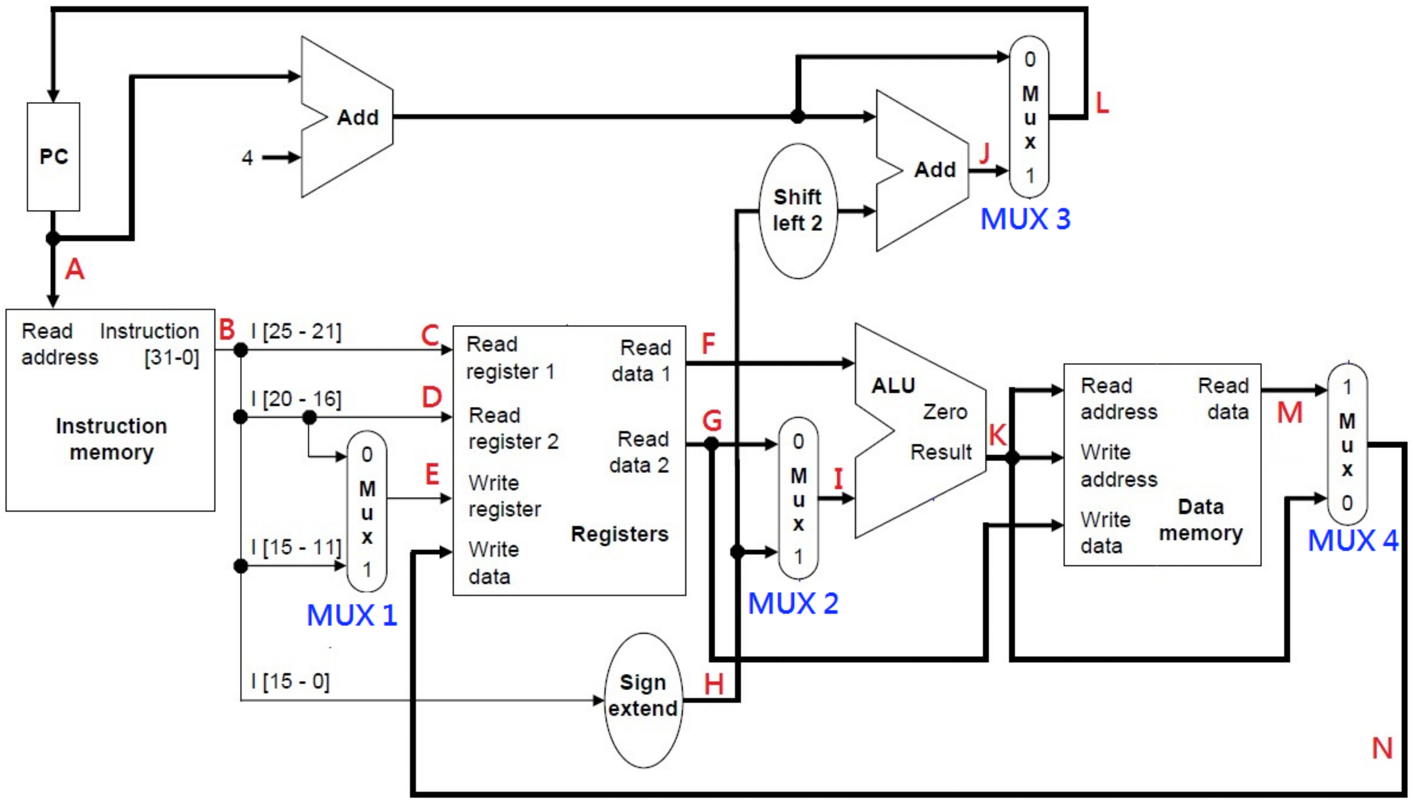
L = 32-bit, PC+4

regwrite = 1

regdest (MUX 1) = 1

branch (MUX 3) = 0

Instruction fetched					
000000	01001	01010	01000	00000	100100
opcode	rs	rt	rd	shamt	func



Exercise 1

- **Trace the execution of the following instructions, figure out the values for label A-N on the datapath.**
  - `lw $t0, 8($t1)`
  - `sw $s0, -16($s1)`
  - `addi $t0, $t1, 100`
  - `beq $s1, $s2, label`

Exercise 2

- **Modify the datapath on slide 3 (with least possible changes) to support the execution of an I-type instruction** `lui $t0, 100`

Setting of Control Signals (Cont'd)

- Setting of control lines (output of control unit):

Instruction	Reg-Dst	ALU-Src	Mem-toReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

`sw` & `beq` will not modify any register, it is ensured by making RegWrite to 0
So, we don't care what write register & write data are

- Input to control unit (i.e. opcode determines setting of control lines):

Instruction	Opcode in decimal	Opcode in binary					
		Op5	Op4	Op3	Op2	Op1	Op0
R-format	0	0	0	0	0	0	0
`lw`	35	1	0	0	0	1	1
`sw`	43	1	0	1	0	1	1
`beq`	4	0	0	0	1	0	0

Summary of Execution Steps

	Actions for instructions			
Step name	**R-type**	**Memory references**	**Branches**	**Jumps**
Instruction fetch	Instruction Fetch			
Instruction decode / register fetch	Instruction decode, read registers			
Execution, addr comp., branch/jump completion	ALU result = RS op RT	ALU result = RS + sign-extend immediate	If (RS == RT) jump	PC = PC[31-28] || 26 bits shift left 2
Memory access, R-type completion		Load: get data from memory		
		Store: RT to memory		
Write back	RD= ALU result	Load: Update RT		

- PC can be potentially modified at either step 1 or 3
- RS - value of RS
- RT – value of RT