MIPS

MIPS Overview

Design and Implementation of MIPS processor supporting the following instructions

- **Memory reference:** `lw`, `sw`
- **Arithmetic/logical:** `add`, `sub`, `and`, `or`, `slt`
- **Control transfer:** `beq`, `j`

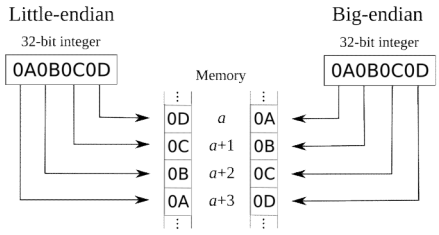
The MIPS ISA has 32 registers. Each register is 32 bits wide

MIPS instructions (add, sub) only operate on registers

Memory is byte addressed
- Each address identifies an 8-bit byte

Words are aligned in memory
- Address must be a multiple of 4

MIPS is Big Endian
- Most-significant byte at least address of a word
- *c.f.* Little Endian: least-significant byte at least address



MIPS Instructions

MIPS R-format instructions

Register numbers

$t0 – $t7 are reg's 8 – 15
$t8 – $t9 are reg's 24 – 25
$s0 – $s7 are reg's 16 – 23

Instruction fields

- op: operation code (opcode)
- rs: first source register number
- rt: second source register number
- rd: destination register number
- shamt: shift amount (00000 for now)
- funct: function code (extends opcode)

Example: add $t0, $s1, $s2

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$0000\ 0010\ 0011\ 0010\ 0100\ 0000\ 0010\ 0000_2$
$= 02324020_{16}$

Note: for sub $t0, $s1, $s2 -- funct = 34

MIPS Instructions

MIPS I-format instructions

Register numbers

$t0 – $t7 are reg's 8 – 15
$t8 – $t9 are reg's 24 – 25
$s0 – $s7 are reg's 16 – 23

- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs

Example: lw $t0, 32($t1)

35	9	8	32

sw $t0, 32($t1)

43	9	8	32

addi $s0, $s0, 4

8	16	16	4

- No subi instruction
  Use addi $s0, $s0, -1

Shift Operations

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Register numbers
$t0 – $t7 are reg's 8 – 15
$t8 – $t9 are reg's 24 – 25
$s0 – $s7 are reg's 16 – 23

- shamt: how many positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - sll by *i* bits multiplies by 2^i
- Shift right logical
  - Shift right and fill with 0 bits
  - srl by *i* bits divides by 2^i (unsigned only)

s0 = 0000 0000 0000 0000 0000 0000 0000 1101
sll $t2, $s0, 8

0	0	16	8	8	0

t2 = 0000 0000 0000 0000 0000 1101 0000 0000

srl $s0, $t1, 10

0	0	9	16	10	2

Conditional Operations

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- beq rs, rt, L1
  - if (rs == rt) branch to instruction labeled L1;
- bne rs, rt, L1
  - if (rs != rt) branch to instruction labeled L1;

Register numbers
$t0 – $t7 are reg's 8 – 15
$t8 – $t9 are reg's 24 – 25
$s0 – $s7 are reg's 16 – 23

```
Loop:sll $t1,$s3,2
     add  $t1,$t1,$s6
     lw   $t0,0($t1)
     bne  $t0,$s5, Exit
     addi $s3,$s3,1
     j    Loop
Exit:
```

Example: beq $t0, $s0, 100

4	8	16	25

16

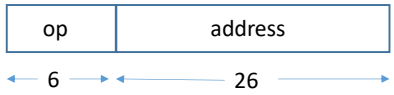
Address: PC + 4 + (25×4)

Note: For bne op = 5

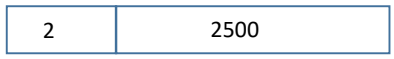
Unconditional Jumps

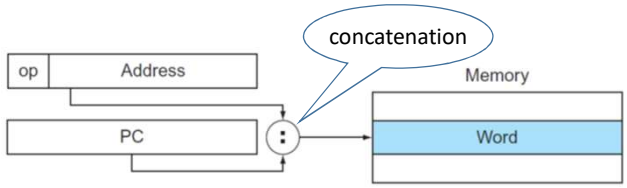
- j L1
  - unconditional jump to instruction labeled L1
- jr reg

J type Instruction



Example: J 10000



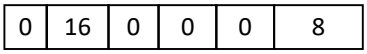


Target address = $PC_{31...28}$: (address × 4)

Register numbers

$t0 – $t7 are reg's 8 – 15
$t8 – $t9 are reg's 24 – 25
$s0 – $s7 are reg's 16 – 23

Example: jr $s0



Example

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- beq rs, rt, L1 -- if (rs == rt) branch to instruction labeled L1;
- bne rs, rt, L1 -- if (rs != rt) branch to instruction labeled L1;
- j L1 -- unconditional jump to instruction labeled L1

C code: f = (i ==j)? g + h : g – h;
- f, g, … in $s0, $s1, …

Compiled MIPS code:

```
    bne $s3, $s4, Else
    add $s0, $s1, $s2
    j   Exit
Else: sub $s0, $s1, $s2
Exit: …
```

C code: while (save[i] == k) i += 1;
- i in $s3, k in $s5, address of save in $s6

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	. . .					

More Conditional Operations

- Set result to 1 if a condition is true
  - Otherwise, set to 0
- slt rd, rs, rt
  - if (rs < rt) rd = 1; else rd = 0;
- slti rt, rs, constant
  - if (rs < constant) rt = 1; else rt = 0;
- Use in combination with beq, bne

```
slt $t0, $s1, $s2    # if ($s1 < $s2)
bne $t0, $zero, L  # branch to L
```

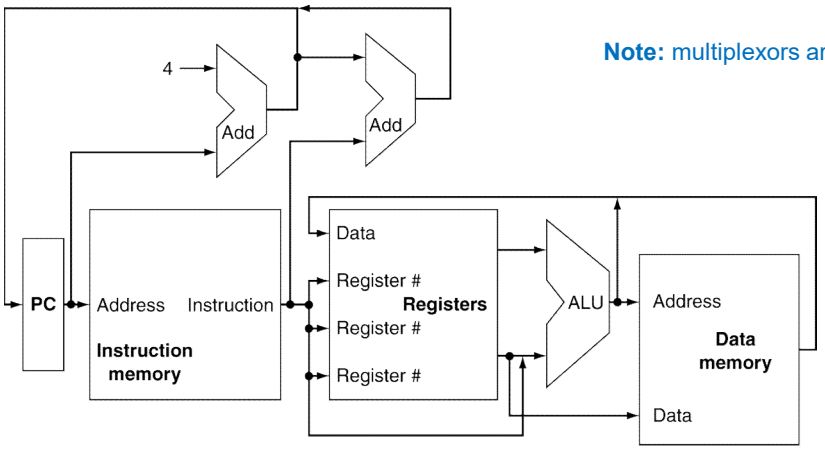
  - Signed comparison: slt, slti
  - Unsigned comparison: sltu, sltui

- Why not blt, bge, etc?
- Hardware for <, ≥, … slower than =, ≠
  - Combining with branch involves more work per instruction, requiring a slower clock
  - All instructions penalized!
- beq and bne are the common case
- This is a good design compromise
- Can we design blt, bge using slt and beq/bne?

Example

$s0 = 1111 1111 1111 1111 1111 1111 1111 1111
$s1 = 0000 0000 0000 0000 0000 0000 0000 0001
slt $t0, $s0, $s1 # signed
–1 < +1 ⇒ $t0 = 1
sltu $t0, $s0, $s1 # unsigned
+4,294,967,295 > +1 ⇒ $t0 = 0

MIPS (Single Cycle) Overview

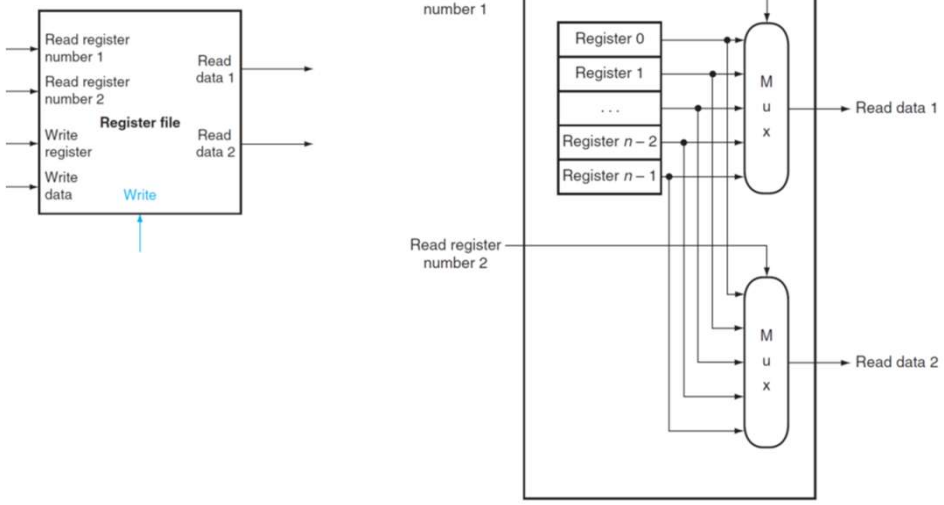


Note: multiplexors are not shown

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

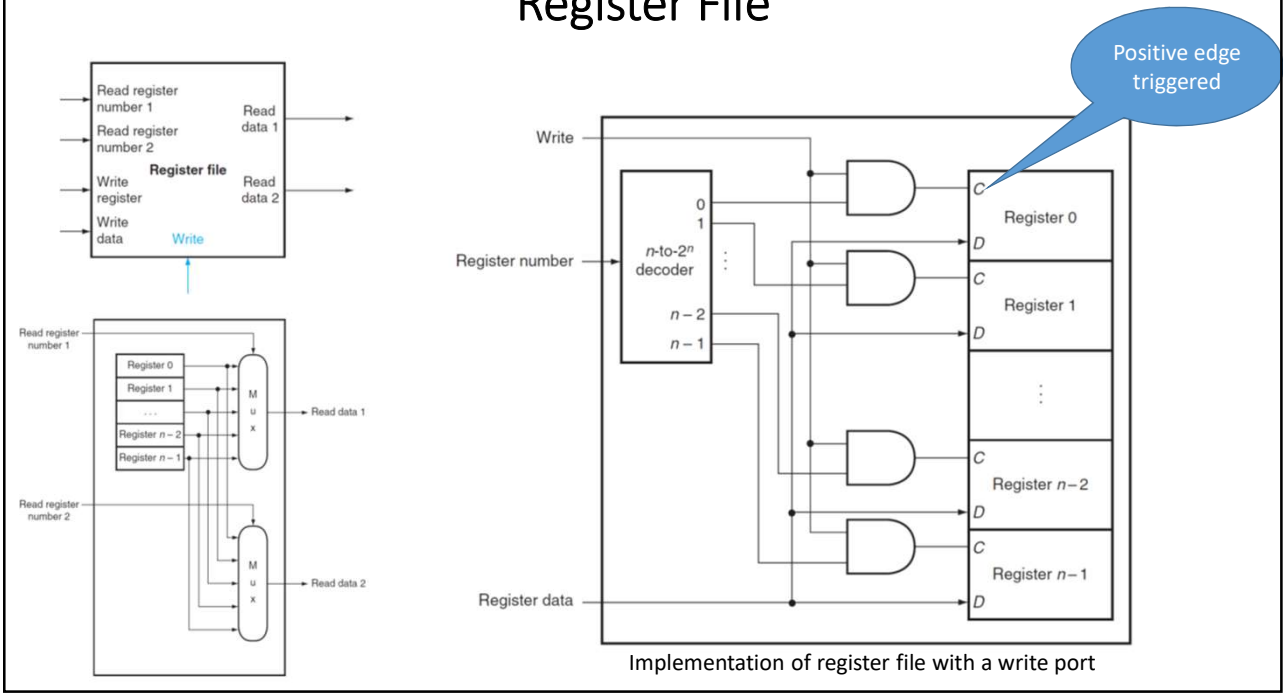
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock? Keep in mind that the latched value remains there for an entire cycle

Register File



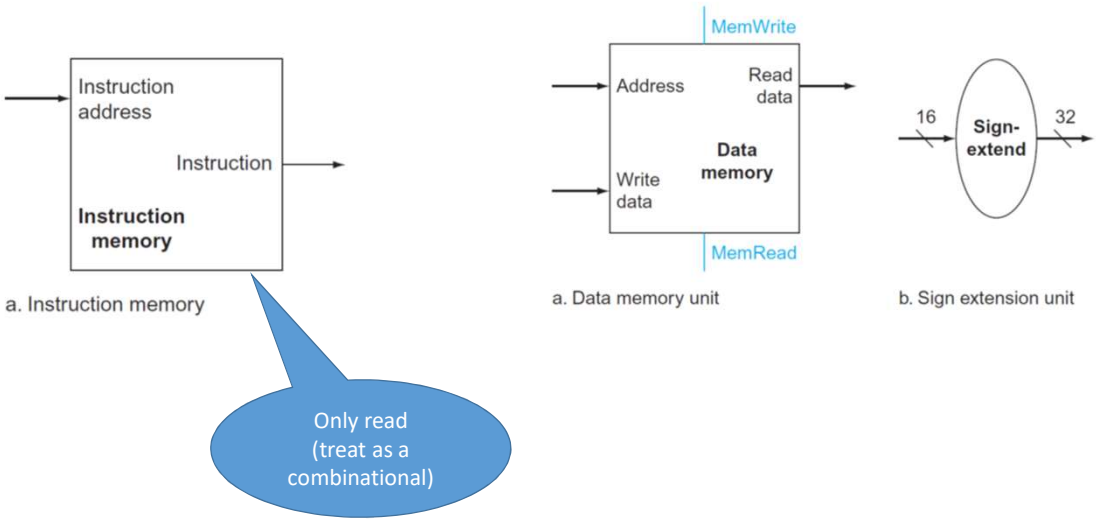
Implementation of register file with two read ports

Register File



Implementation of register file with a write port

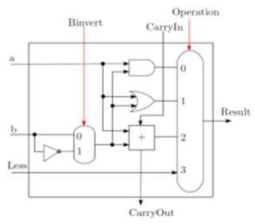
Data Memory Units



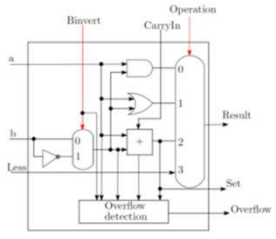
ALU

Supporting conditional branch: beq rs, rt, L1 -- if (rs == rt) branch to instruction labeled L1

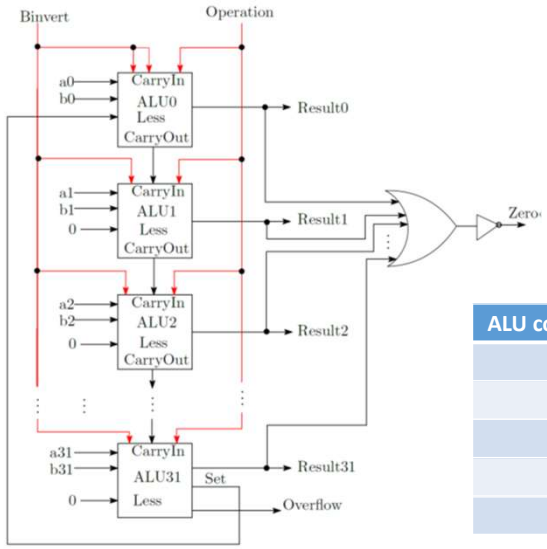
Idea: (a-b) = 0 → a = b

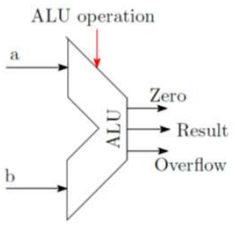


ALU 0 – ALU 30



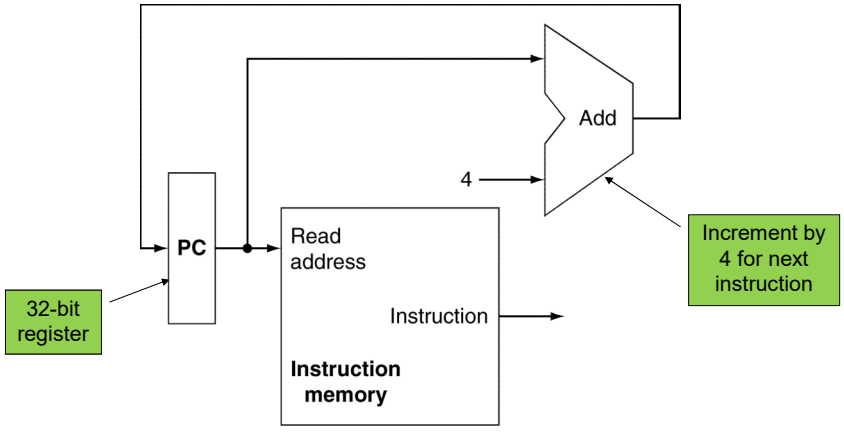
ALU 31



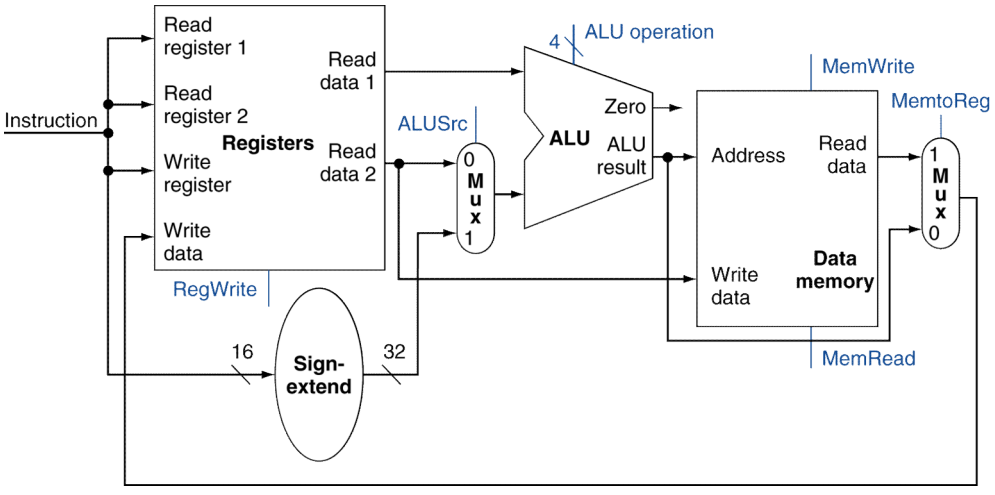


ALU control lines	Function
000	and
001	or
010	add
110	subtract
111	Set on less than

Datapath: Instruction Fetch



Datapath: R-Format and Load/Store



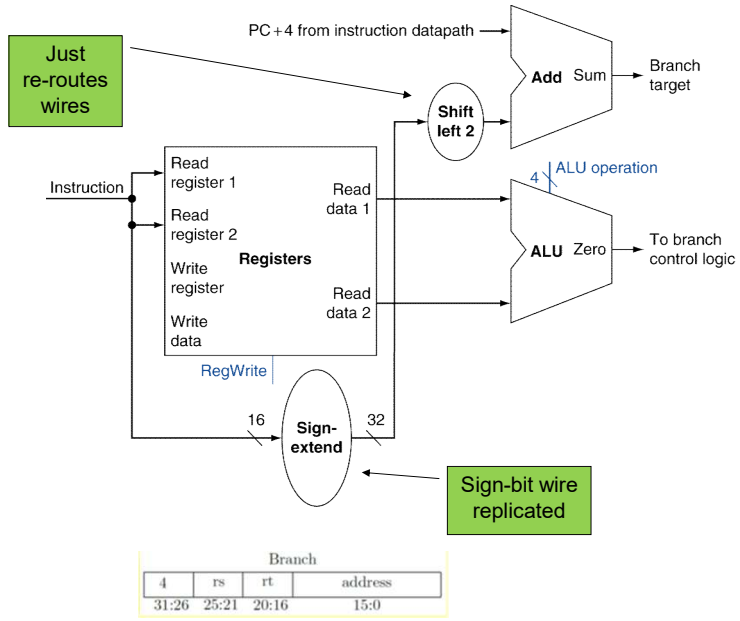
R-type

0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

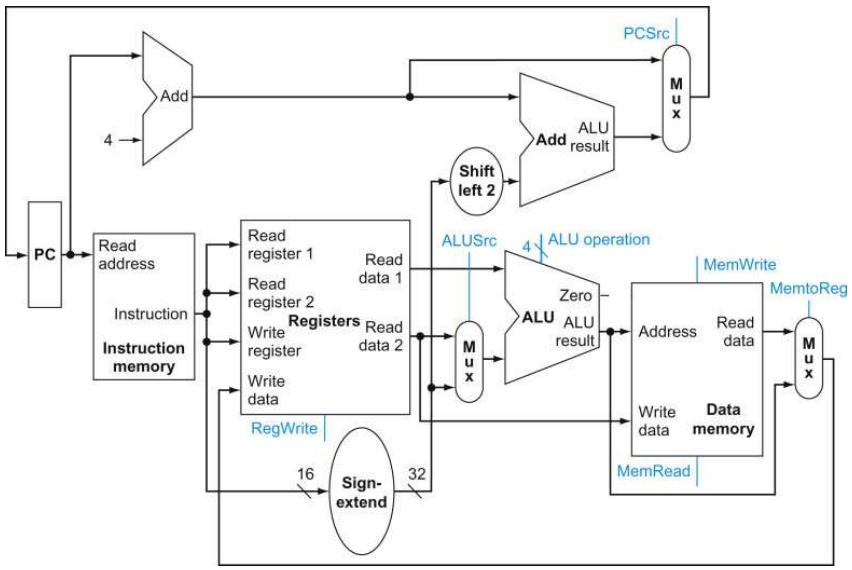
Load/Store

35 or 43	rs	rt	address
31:26	25:21	20:16	15:0

Datapath: Branch



Single Cycle Datapath



Datapath and the Clock

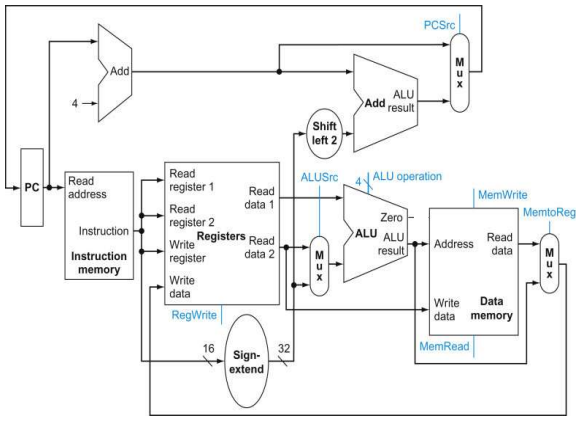
STEP 1: A new instruction is loaded from memory. The control unit sets the datapath signals appropriately so that

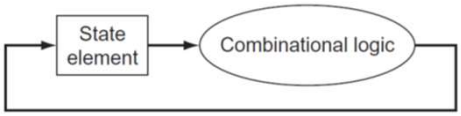
- registers are read,
- ALU output is generated,
- data memory is read and
- branch target addresses are computed.

STEP 2:

- The register file is updated for arithmetic or lw instructions.
- Data memory is written for a sw instruction.
- The PC is updated to point to the next instruction.

In a single-cycle datapath everything in STEP 1 must complete within one clock cycle.





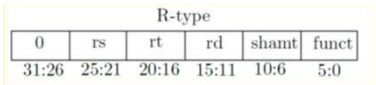
ALU Control

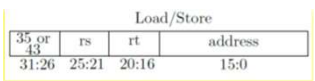
- ALU used for
  - Load/Store: F = add
  - Branch: F = subtract
  - R-type: F depends on funct field

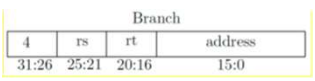
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

- 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

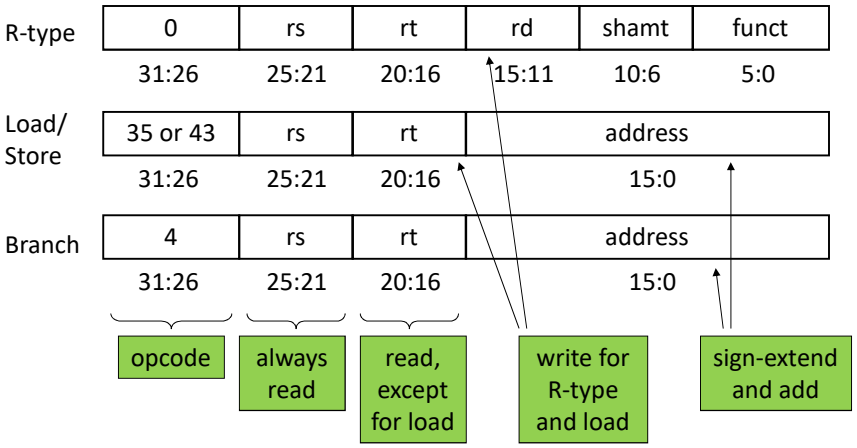




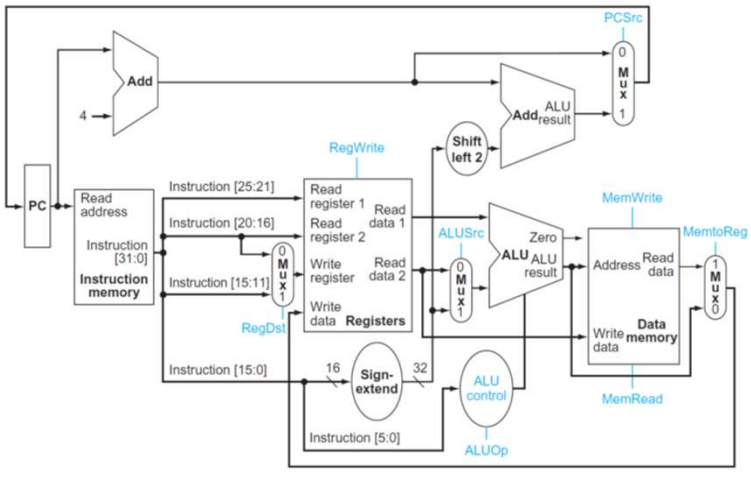


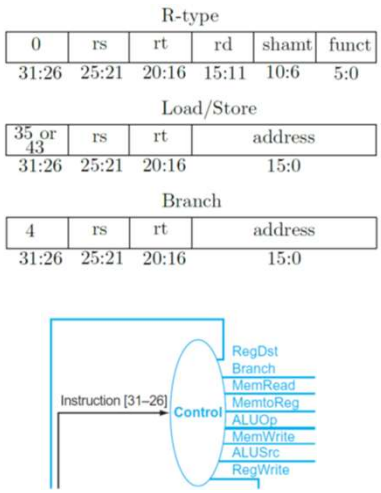
The Main Control Unit

- Control signals derived from instruction



Single Cycle Datapath With Control

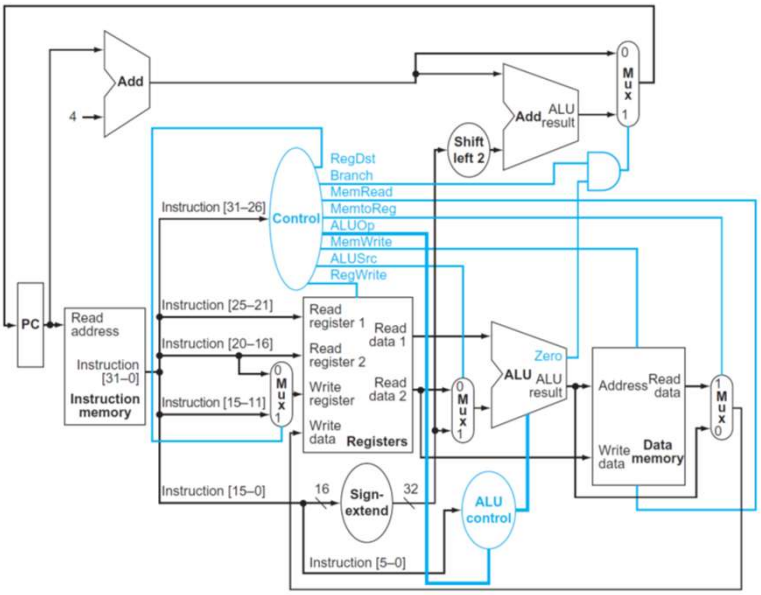




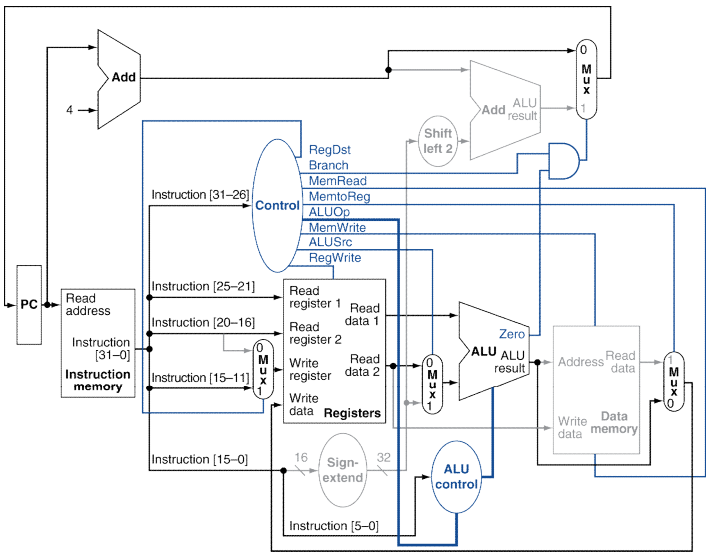
Q. How to augment datapath for conditional jump instruction e.g., beq?

Hint: What should be the control logic for PCSrc?

Single Cycle Datapath With Control

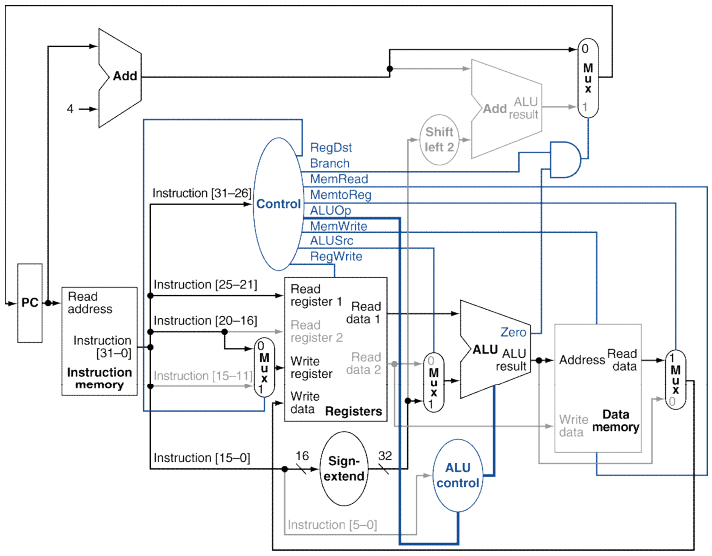


R-Type Instruction



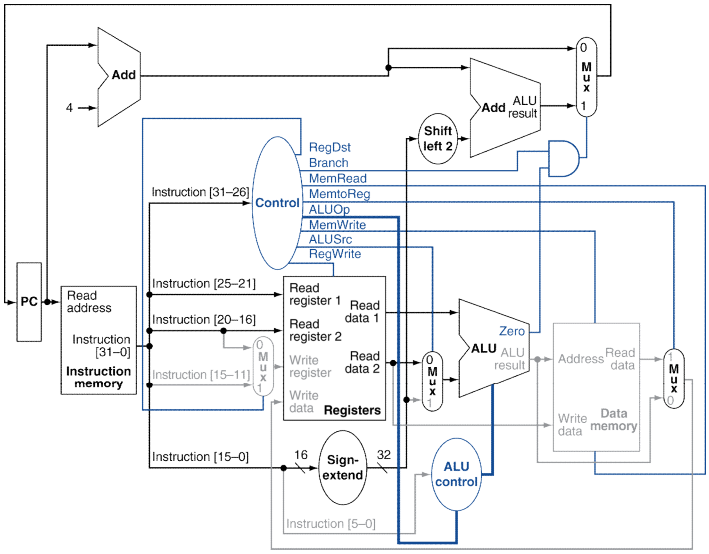
Input or output	Signal name	R-format
Inputs	Op5	0
	Op4	0
	Op3	0
	Op2	0
	Op1	0
	Op0	0
Outputs	RegDst	1
	ALUSrc	0
	MemtoReg	0
	RegWrite	1
	MemRead	0
	MemWrite	0
	Branch	0
	ALUOp1	1
	ALUOp0	0

Store and Load Instruction



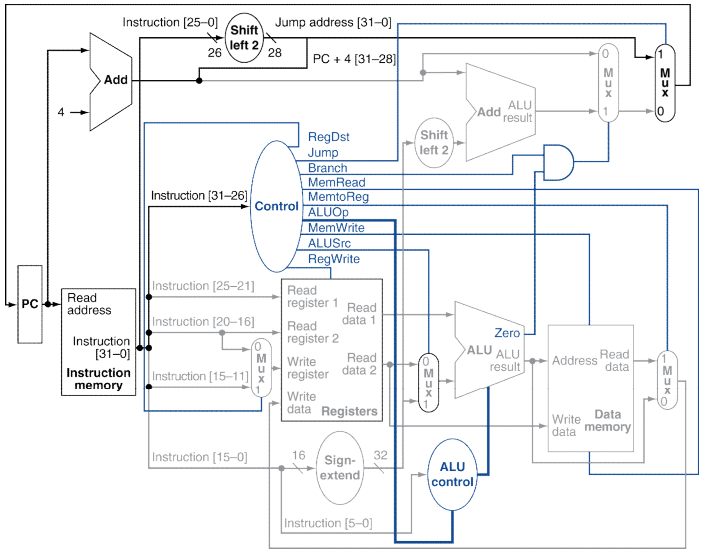
Input or output	Signal name	lw	sw
Inputs	Op5	1	1
	Op4	0	0
	Op3	0	1
	Op2	0	0
	Op1	1	1
	Op0	1	1
Outputs	RegDst	0	X
	ALUSrc	1	1
	MemtoReg	1	X
	RegWrite	1	0
	MemRead	1	0
	MemWrite	0	1
	Branch	0	0
	ALUOp1	0	0
	ALUOp0	0	0

Branch-on-Equal Instruction



Input or output	Signal name	beq
Inputs	Op5	0
	Op4	0
	Op3	0
	Op2	1
	Op1	0
	Op0	0
Outputs	RegDst	X
	ALUSrc	0
	MemtoReg	X
	RegWrite	0
	MemRead	0
	MemWrite	0
	Branch	1
	ALUOp1	0
	ALUOp0	1

Datapath With Unconditional Jumps



2	address
31:26	25:0