Implementation Overview

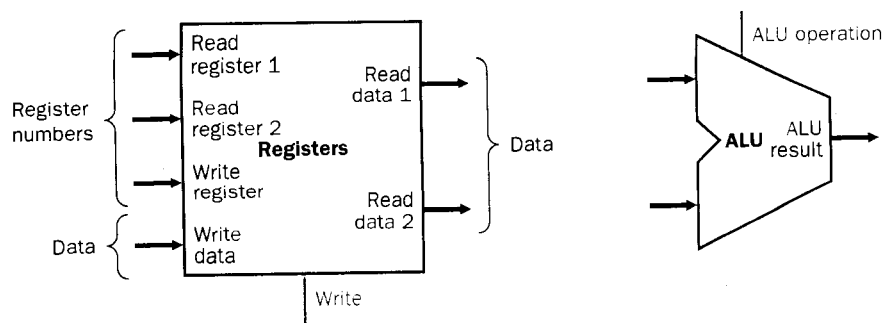
For all instructions:
PC → memory address
memory data → instruction

memory reference: (lw, sw)	compute address (ALU) give address to memory read or write data to memory (and write or read to register)
arithmetic:	fetch values from registers ALU write result
branch, (jump):	~~if true~~ test for equal/etc. (ALU) compute dest addr ~~(ALU)~~ go there

Simplicity & Regularity of Instructions make our lives easier.

Necessary components

- State – mem, PC
- No state – ALU



a. Registers

b. ALU

Abstract View of MIPS Implementation

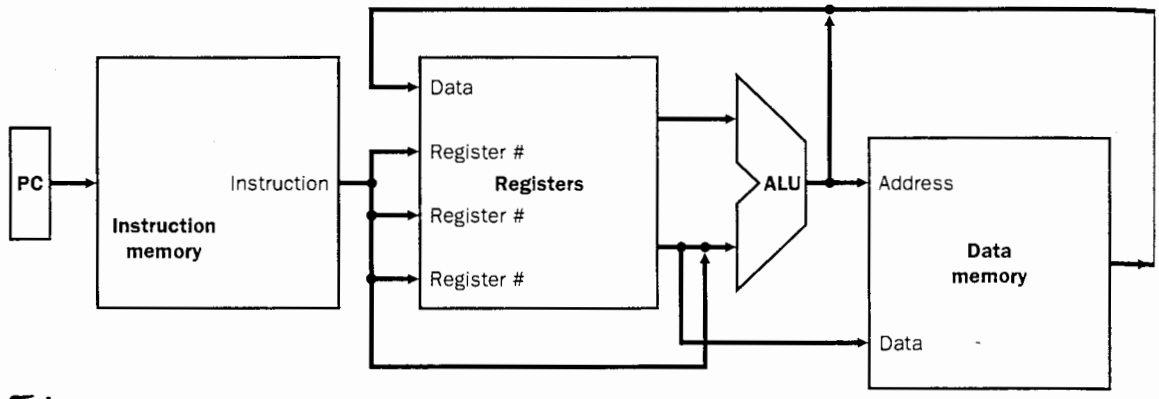
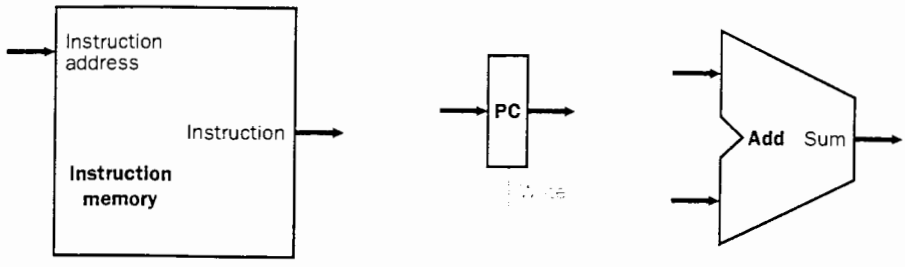


Fig 5.1

Some Necessary Parts



a. Instruction memory b. Program counter c. Adder

Fig 5.5

Arithmetic and Logical Instructions

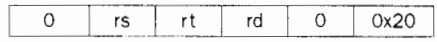
Absolute value

```
abs rdest, rsrc                pseudoinstruction
```

Put the absolute value of register `rsrc` in register `rdest`.

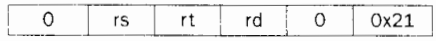
Addition (with overflow)

```
add rd, rs, rt
```



Addition (without overflow)

```
addu rd, rs, rt
```



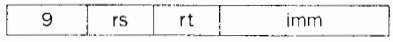
Put the sum of registers `rs` and `rt` into register `rd`.

Addition immediate (with overflow)

```
addi rt, rs, imm
```

Addition immediate (without overflow)

```
addiu rt, rs, imm
```



Put the sum of register `rs` and the sign-extended immediate into register `rt`.

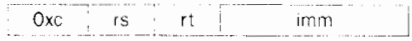
AND

```
and rd, rs, rt
```

Put the logical AND of registers `rs` and `rt` into register `rd`.

AND immediate

```
andi rt, rs, imm
```



Put the logical AND of register `rs` and the zero-extended immediate into register `rt`.

Instruction Fetch & Program Counter Increment

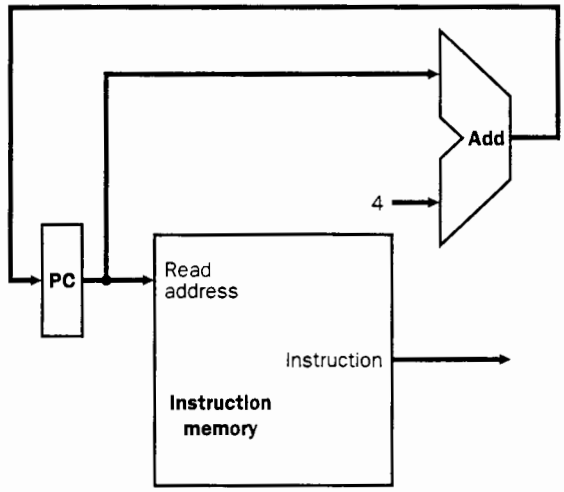


Fig 5.6

Arithmetic (R-Format) instructions need to read two values from registers and write the result back to a register.

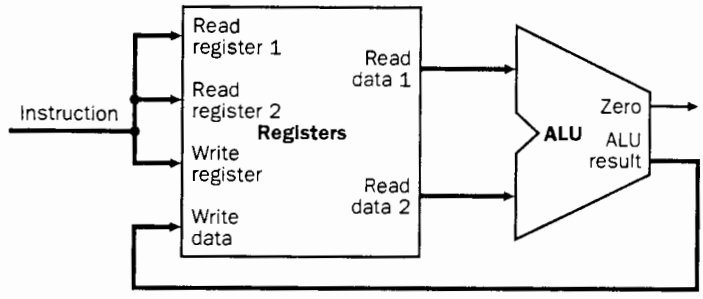
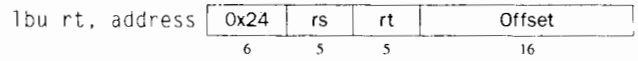


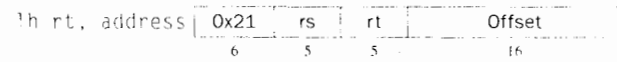
Fig 5.8

Load unsigned byte

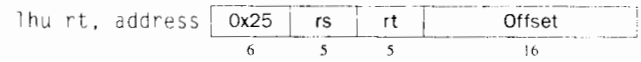


Load the byte at *address* into register `rt`. The byte is sign-extended by `lb`, but not by `lbu`.

Load halfword

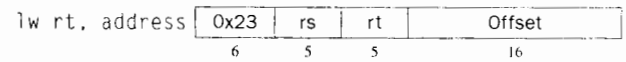


Load unsigned halfword



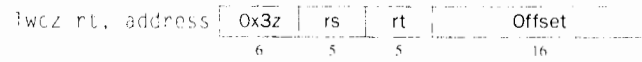
Load the 16-bit quantity (halfword) at *address* into register `rt`. The halfword is sign-extended by `lh`, but not by `lhu`.

Load word



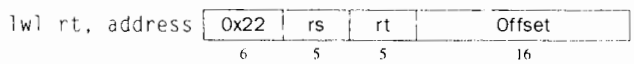
Load the 32-bit quantity (word) at *address* into register `rt`.

Load word coprocessor

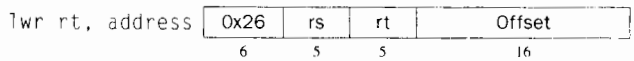


Load the word at *address* into register `rt` of coprocessor `z` (0–3). The floating-point unit is $z = 1$.

Load word left



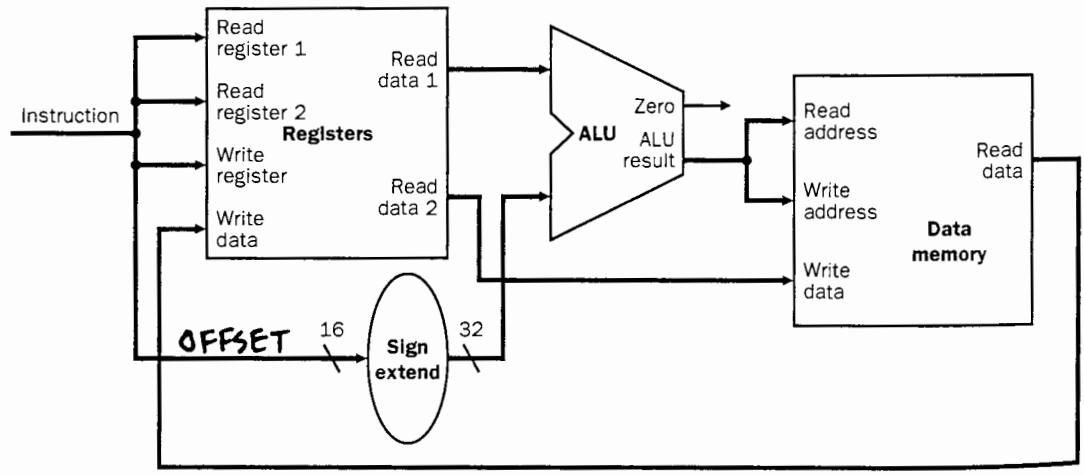
Load word right



Load the left (right) bytes from the word at the possibly unaligned *address* into register `rt`.

LOAD / STORE INSTRUCTIONS

lw $3, 23($4)
- 23 ↑ offset
- ($4) ← base address



5.10

- lw:

  Compute address: Base + Offset
  Fetch value from register
  Write it to memory

- sw:

  Compute address: Base + Offset
  Fetch value from memory
  Write it to register

Branch coprocessor *z* true

```
bczt label
```

0x1*z*	8	1	Offset
6	5	5	16

Branch coprocessor *z* false

```
bczf label
```

0x1*z*	8	0	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if *z*'s condition flag is true (false). *z* is 0, 1, 2, or 3. The floating-point unit is *z* = 1.

Branch on equal

```
beq rs, rt, label
```

4	rs	rt	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register `rs` equals `rt`.

Branch on greater than equal zero

```
bgez rs, label
```

1	rs	1	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register `rs` is greater than or equal to 0.

Branch on greater than equal zero and link

```
bgezal rs, label
```

1	rs	0x11	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register `rs` is greater than or equal to 0. Save the address of the next instruction in register 31.

Branch on greater than zero

```
bgtz rs, label
```

7	rs	0	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register `rs` is greater than 0.

BEQ INSTRUCTION

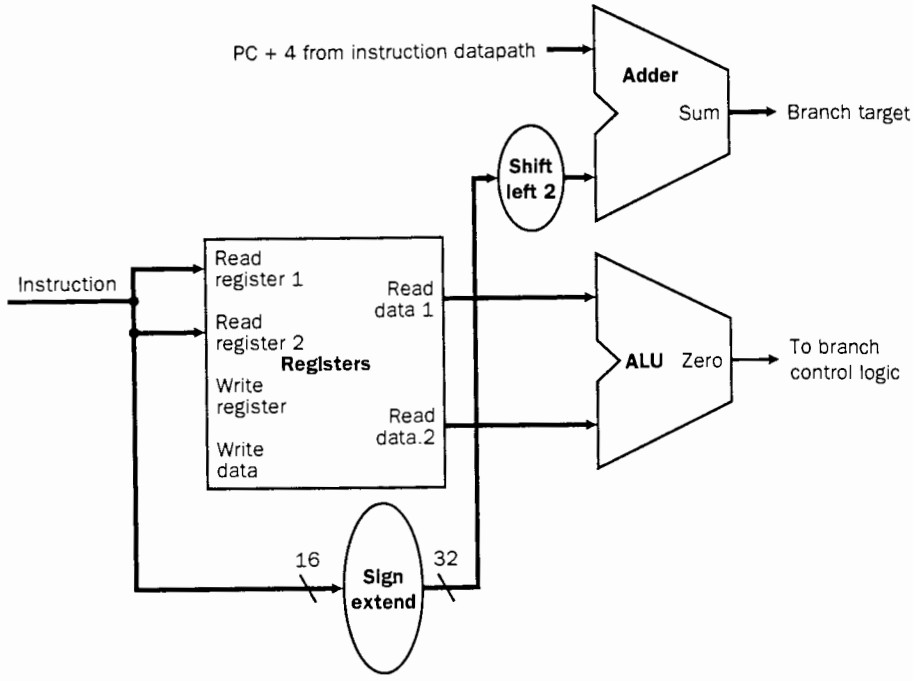
beq $3, $4, offset

① Read $3 and $4

② Are they equal?

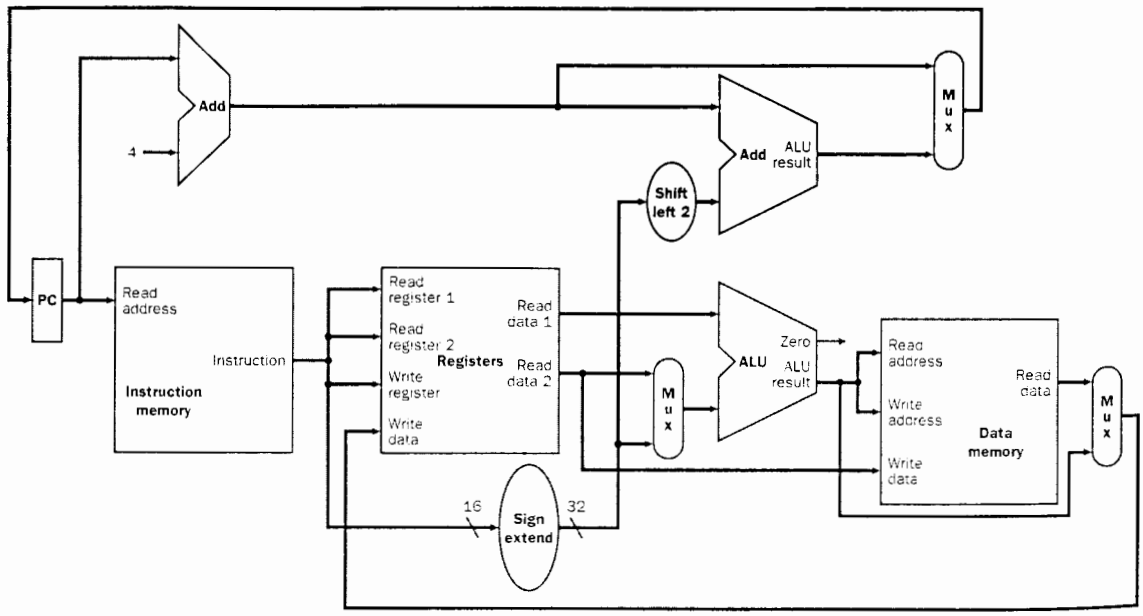
③ Compute new address = PC + offset

complication: need to shift offset by 2



Put it all together – full datapath

- Added some logic to finish making beq work



5.14

But, we still need to add control signals

Add Control Lines

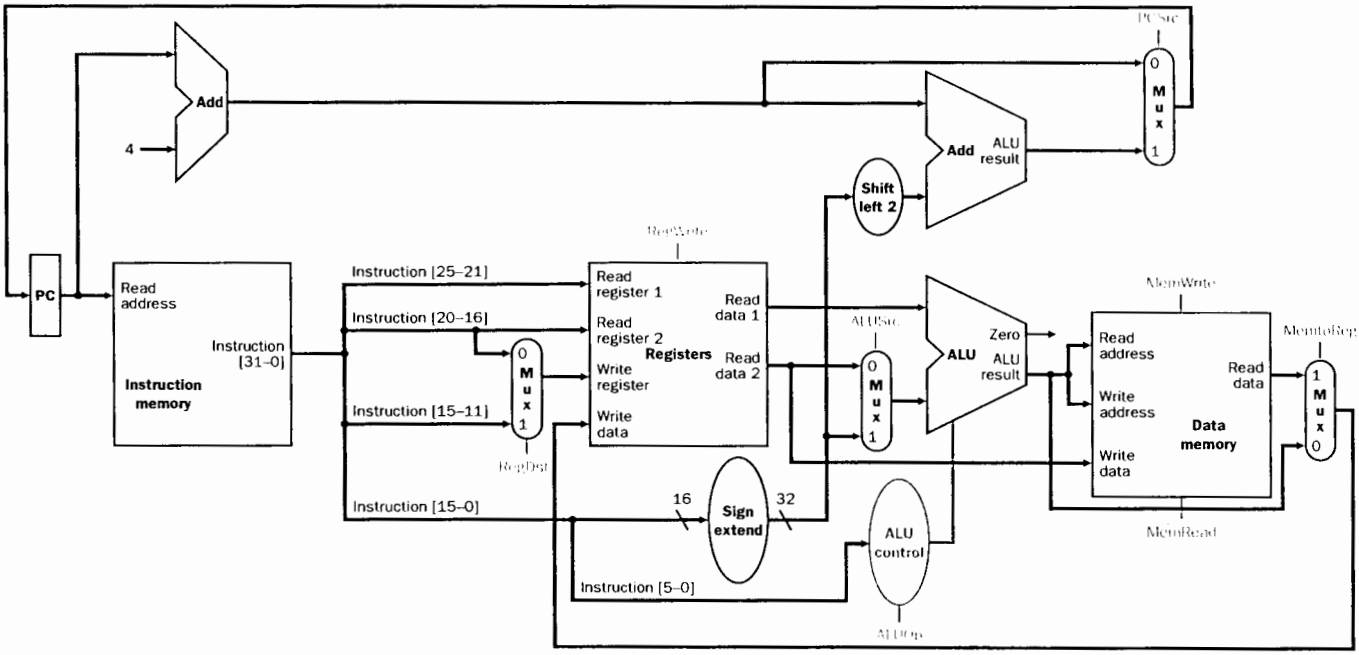
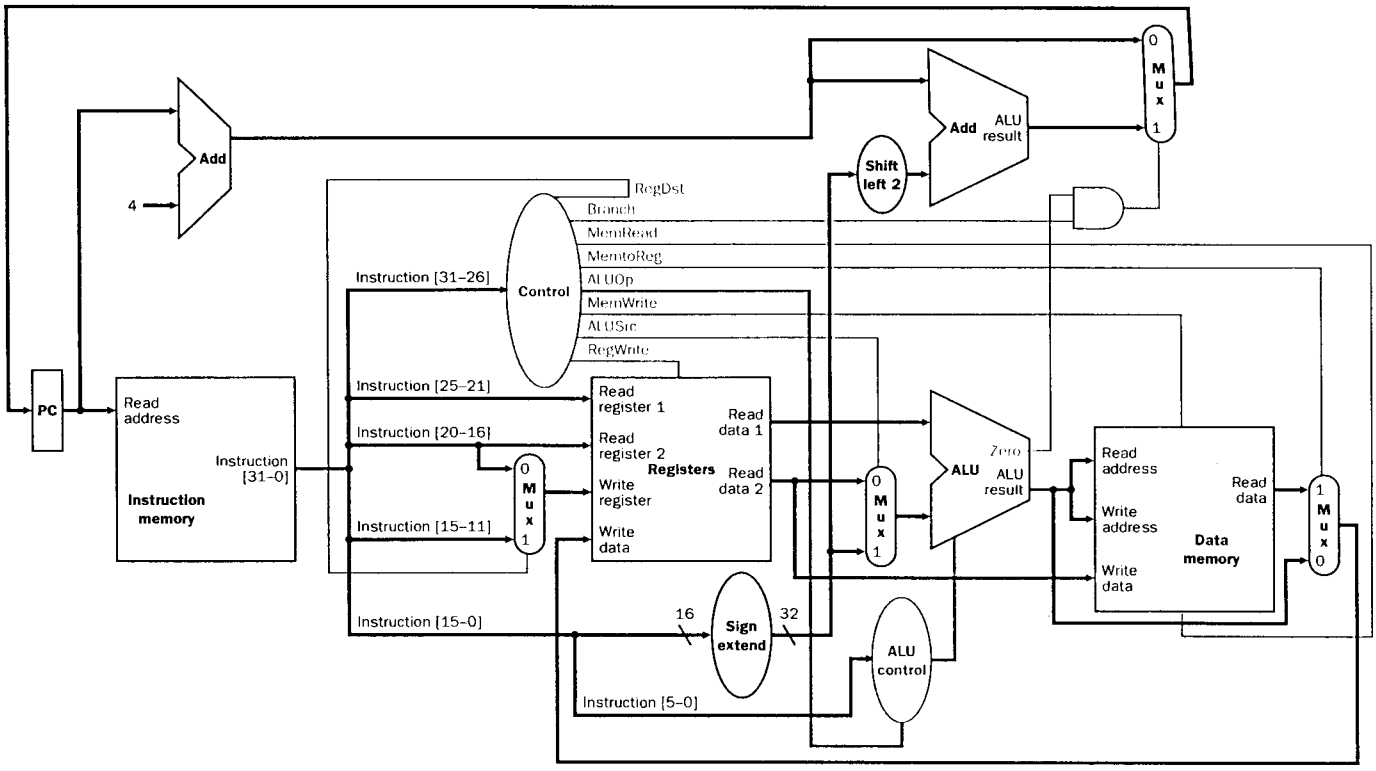


FIGURE 5.20 The datapath of Figure 5.14 with all necessary multiplexors and all control lines identified. The control lines are shown in color. ıe ALU control block has also been added

Add Control Unit



Control unit can be implemented as a PLA.

Field	0	rs	rt	rd	shamt	funct
Bit positions	31–26	25–21	20–16	15–11	10–6	5–0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31–26	25–21	20–16	15–0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31–26	25–21	20–16	15–0

c. Branch instruction