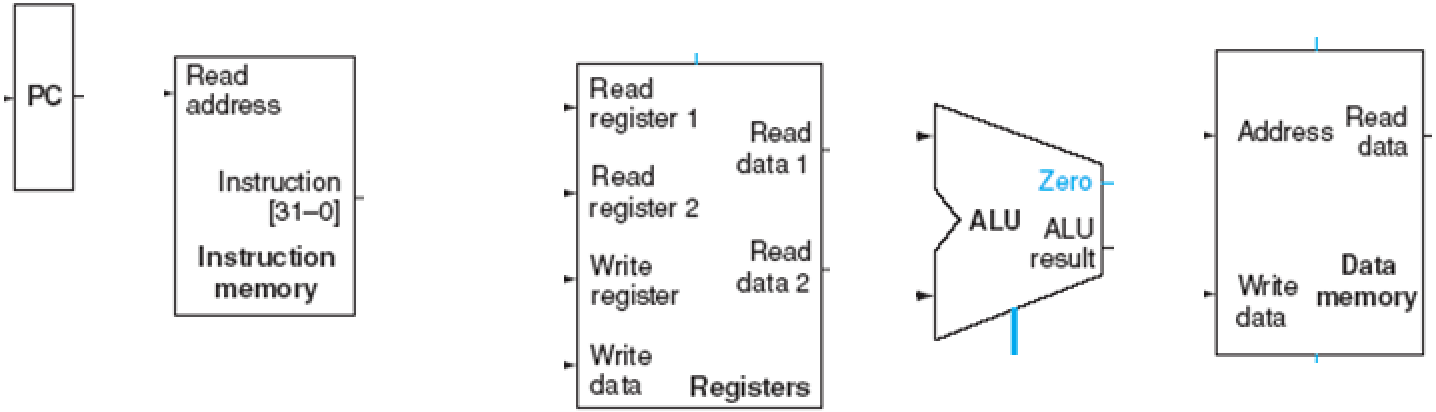
MIPS processor continued

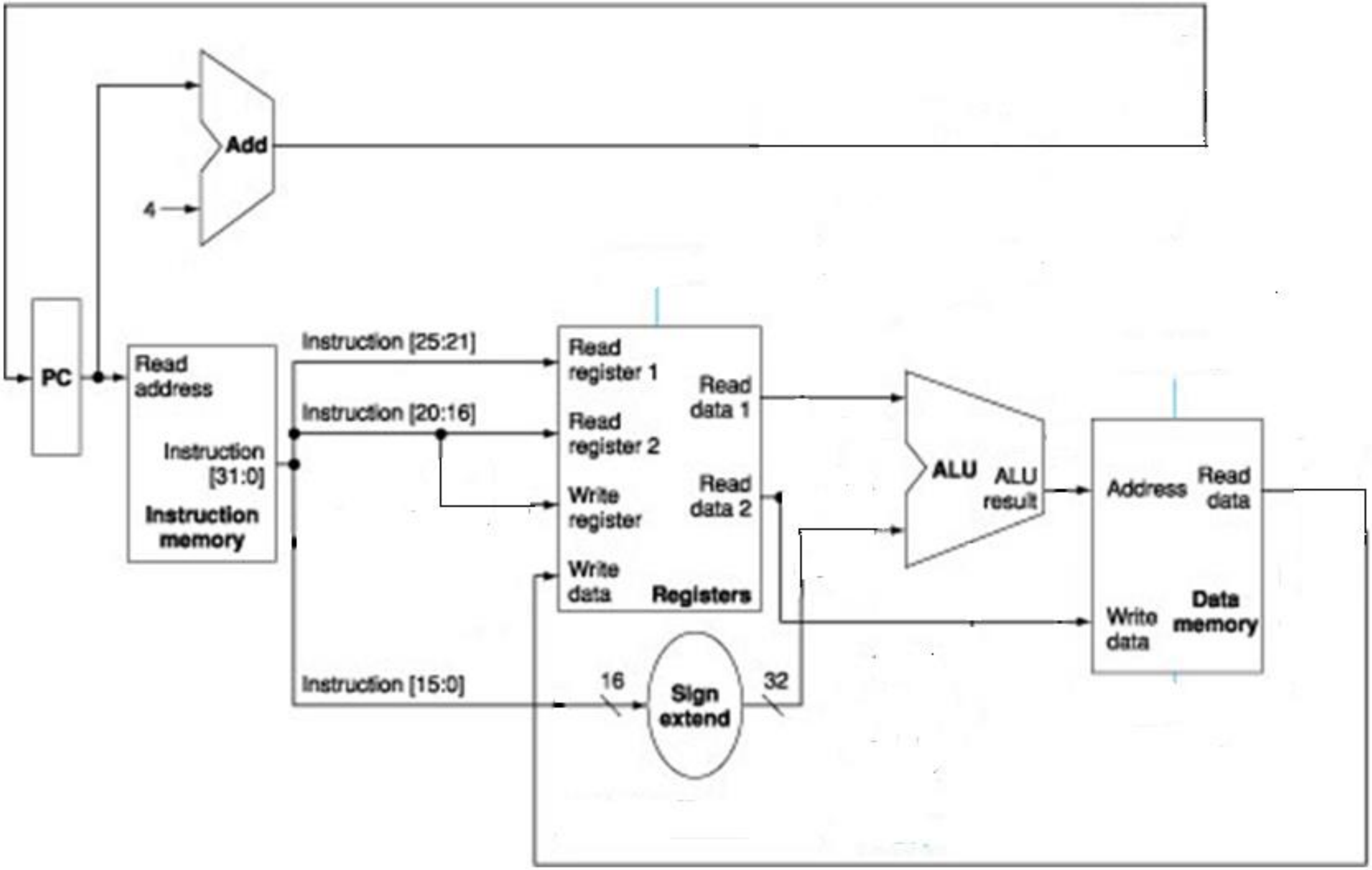
Review

- Different parts in the processor should be connected appropriately to be able to carry out the functions.
- Connections depending on what we need
- Learnt R-type, lw, sw, beq

lw & sw?

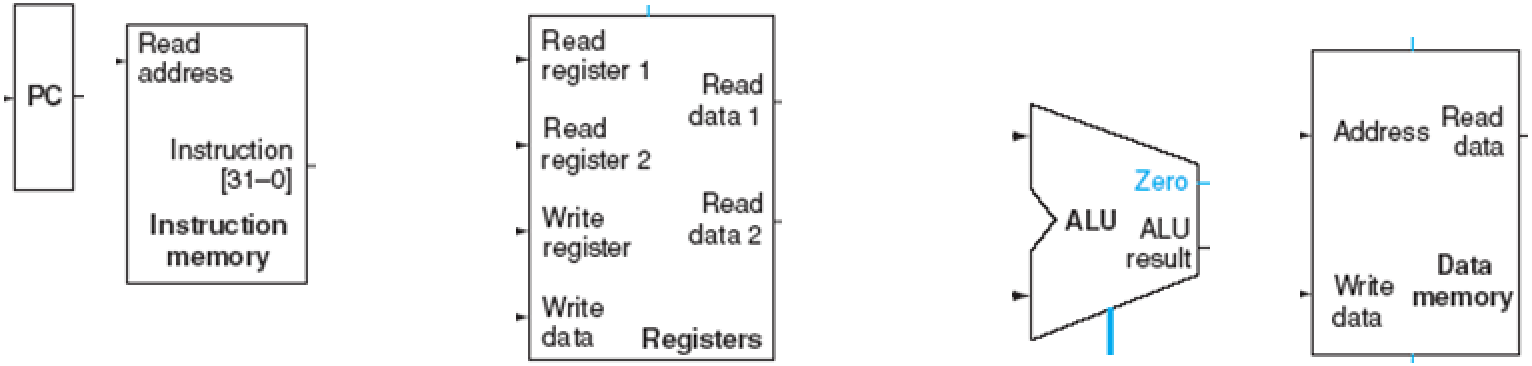

PC
Read address
Instruction [31–0]
Instruction memory
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ALU
Zero
ALU result
Address
Read data
Write data
Data memory

Data path only for lw and sw (answer)



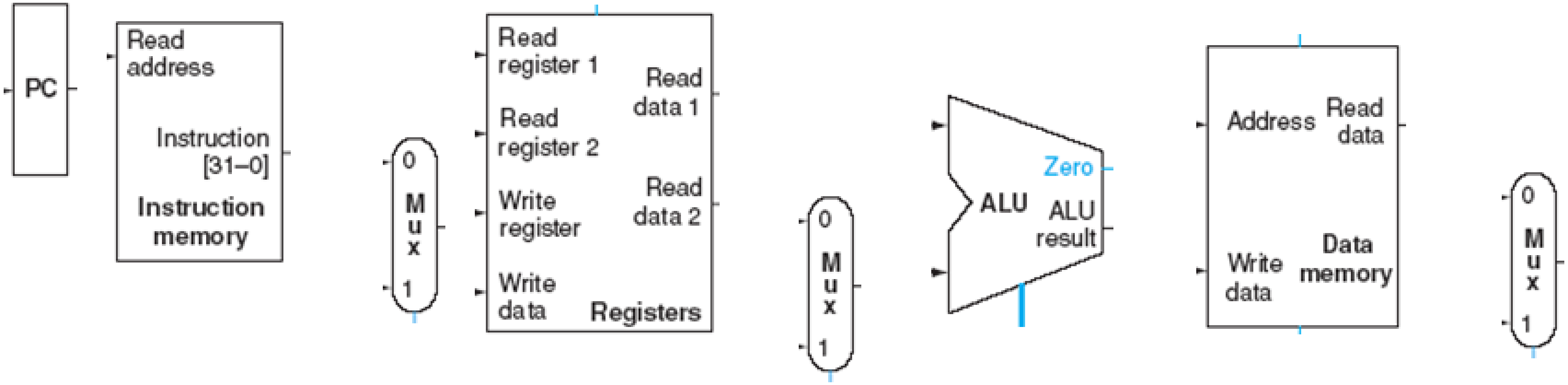
Data path for both R-type and memory-type instructions

add $rd, $rs, $rt, format: opcode (6 bits) rs (5 bits) rt (5 bits) rd (5 bits) 00000 funct (6 bits)
lw $rt, offset_value($rs): opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)
sw $rt, offset_value($rs): opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)

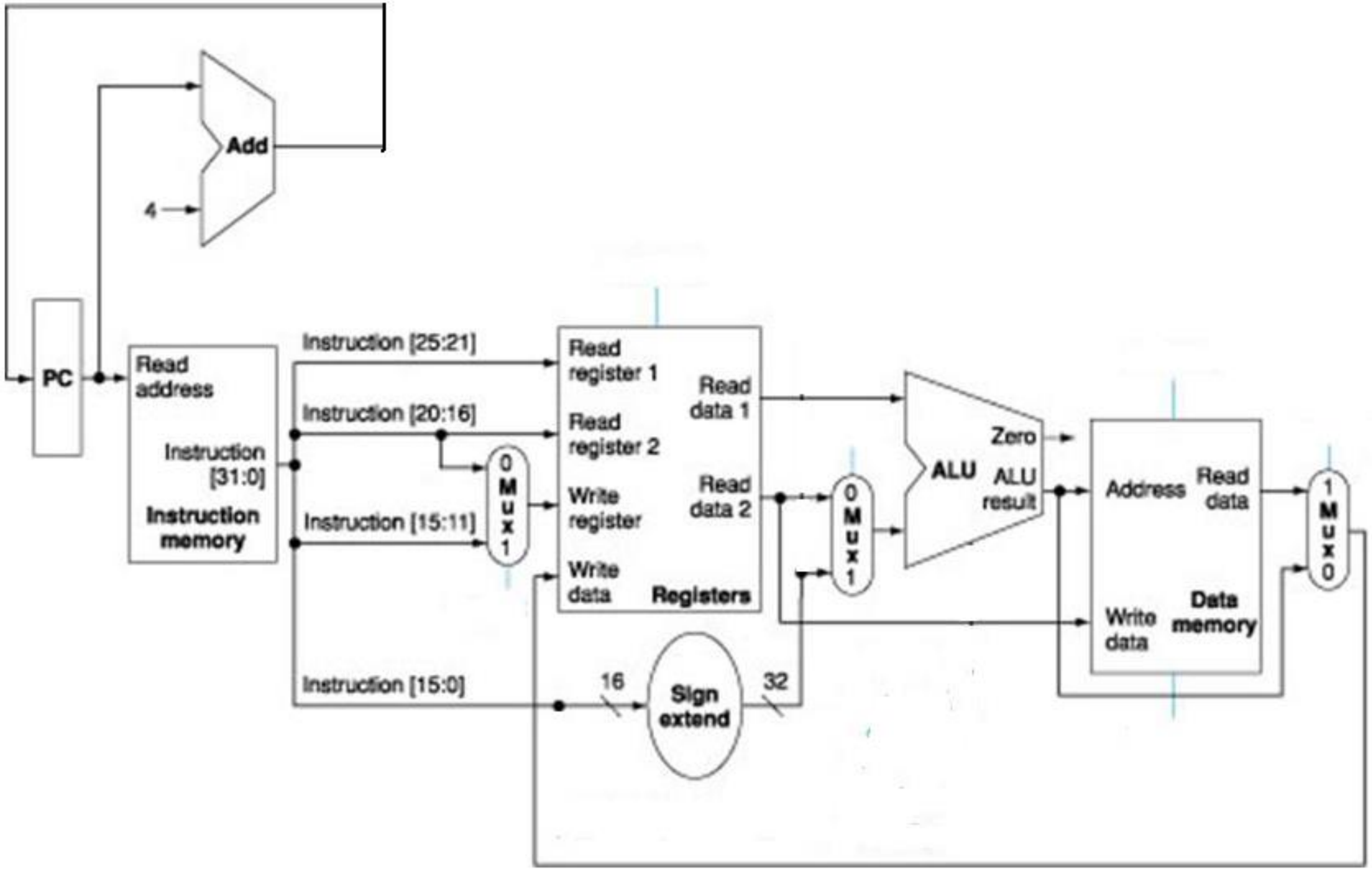


Data path for both R-type and memory-type instructions

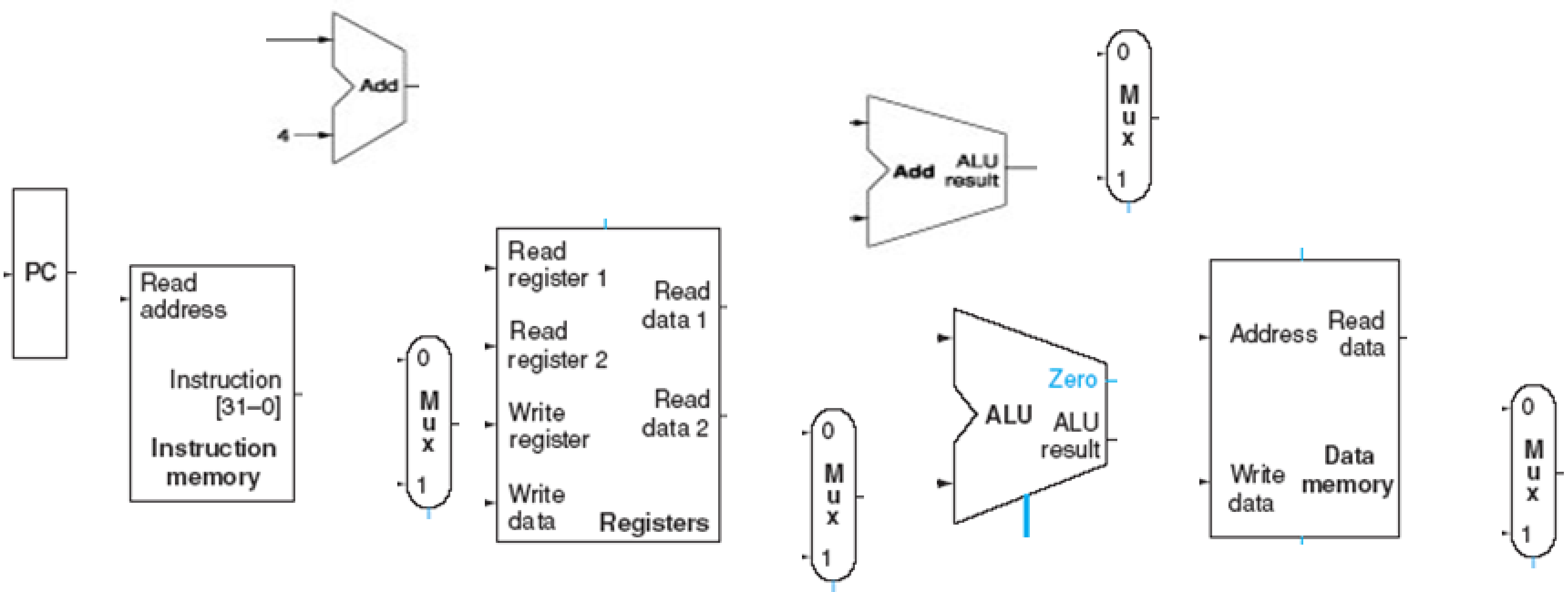
add $rd, $rs, $rt, format: opcode (6 bits) rs (5 bits) rt (5 bits) rd (5 bits) 00000 funct (6 bits)
lw $rt, offset_value($rs): opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)
sw $rt, offset_value($rs): opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)



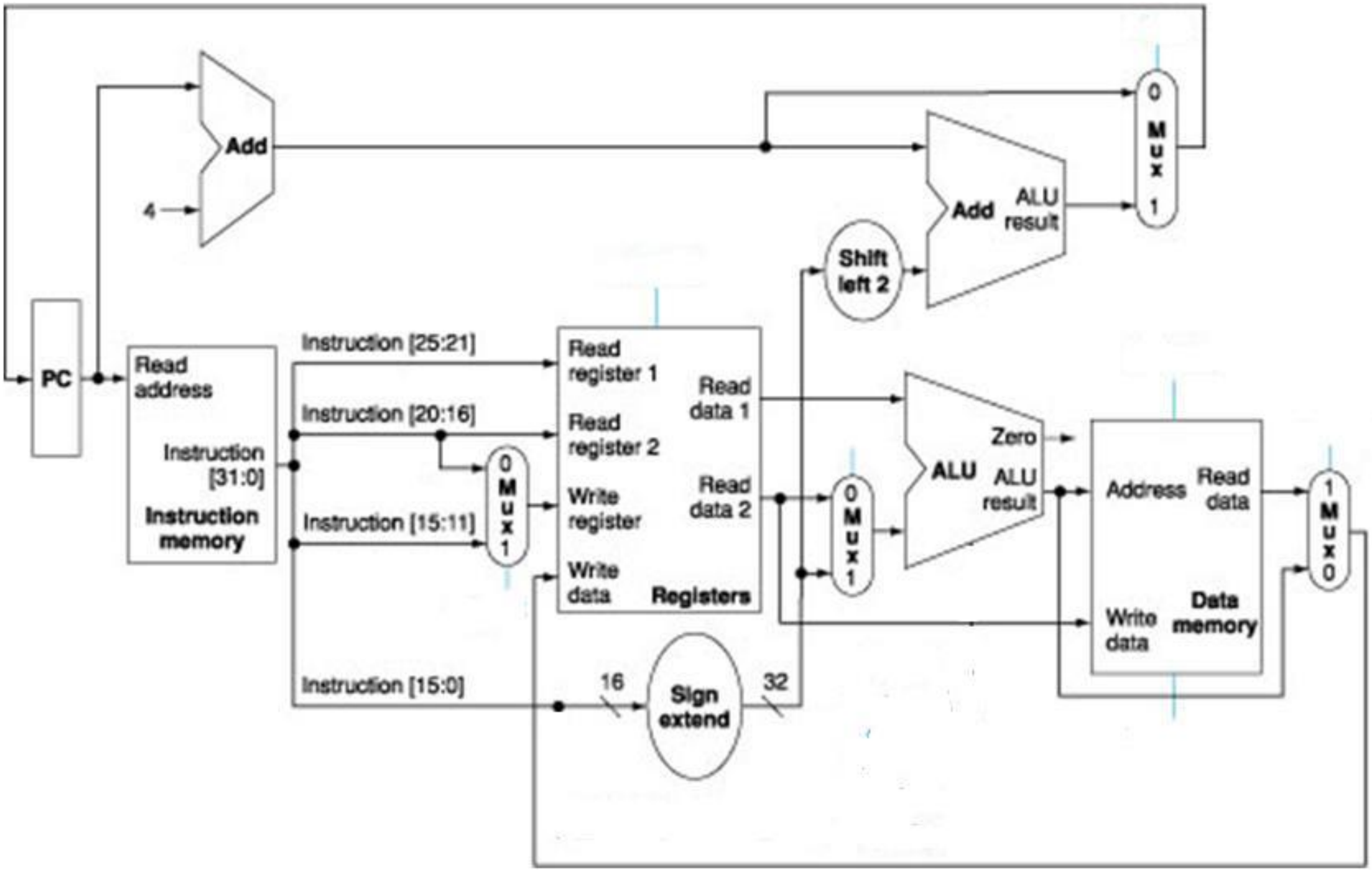
Answer


Add
4
PC
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Instruction [15:0]
0 Mux 1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign extend
32
0 Mux 1
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
1 Mux 0

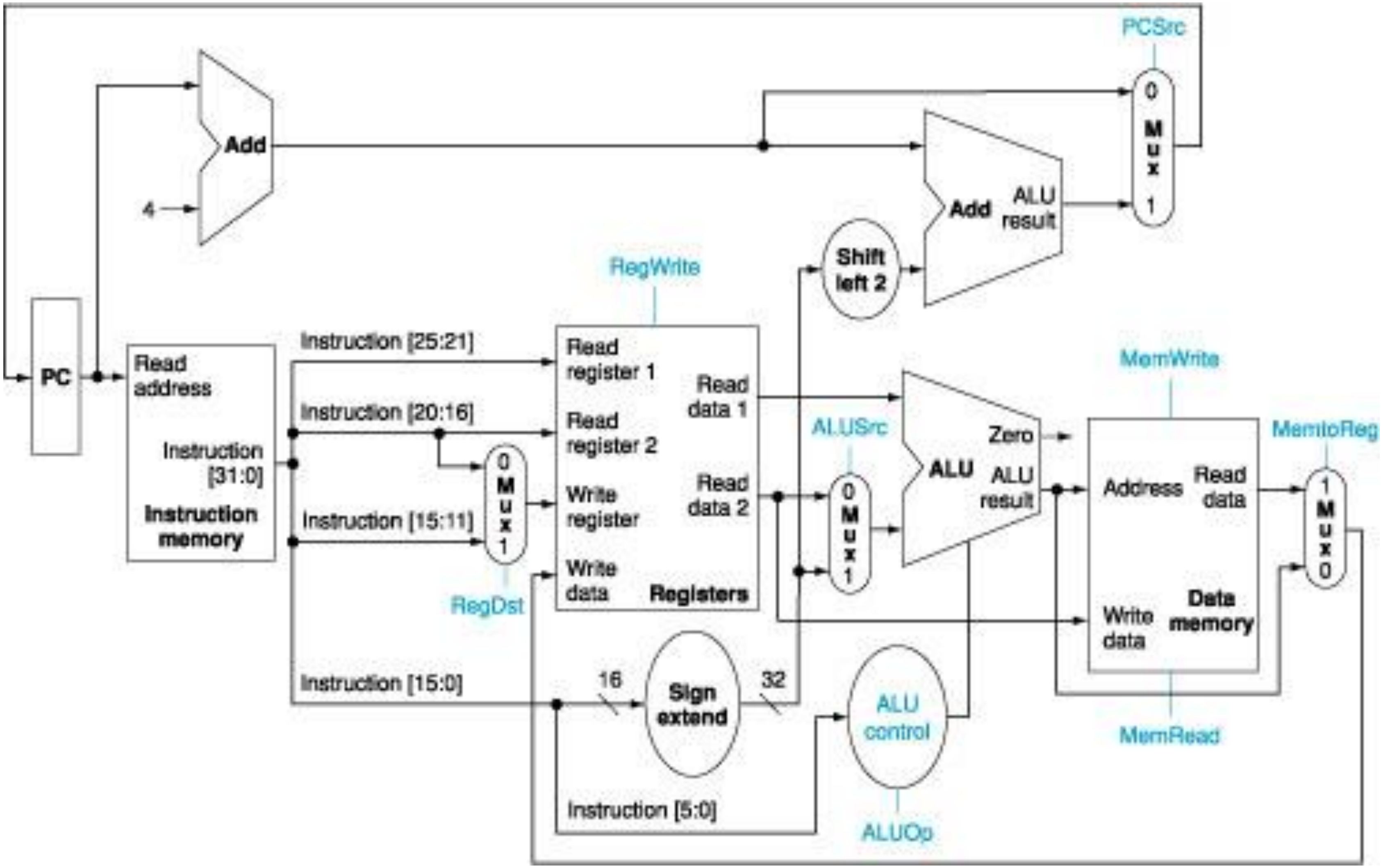
Datapath for R-type, memory, and branch operations



Datapath for R-type, memory, and branch operations (Answer)



Datapath for Memory, R-type and Branch Instructions, plus the control signals

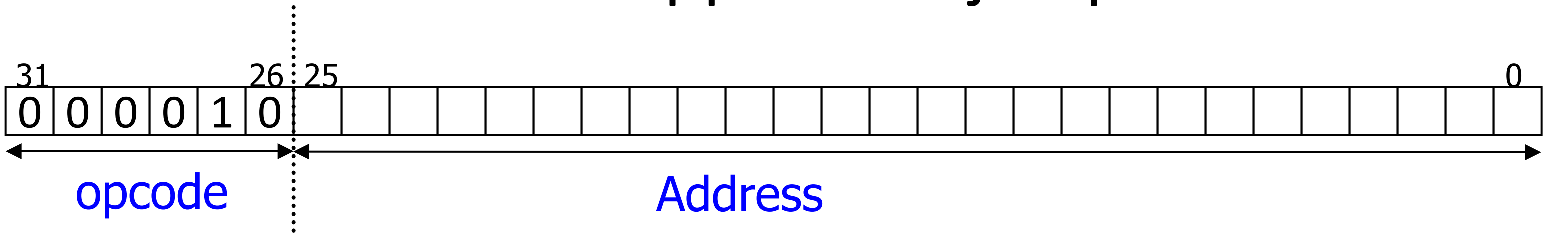


Jump Instruction

- Jump instruction seems easy to implement
  - We just need to replace the lower 28 bits of the PC with the lower 26 bits of the instruction shifted by 2 bits
    - The shift is achieved by simply concatenating 00 to the jump offset

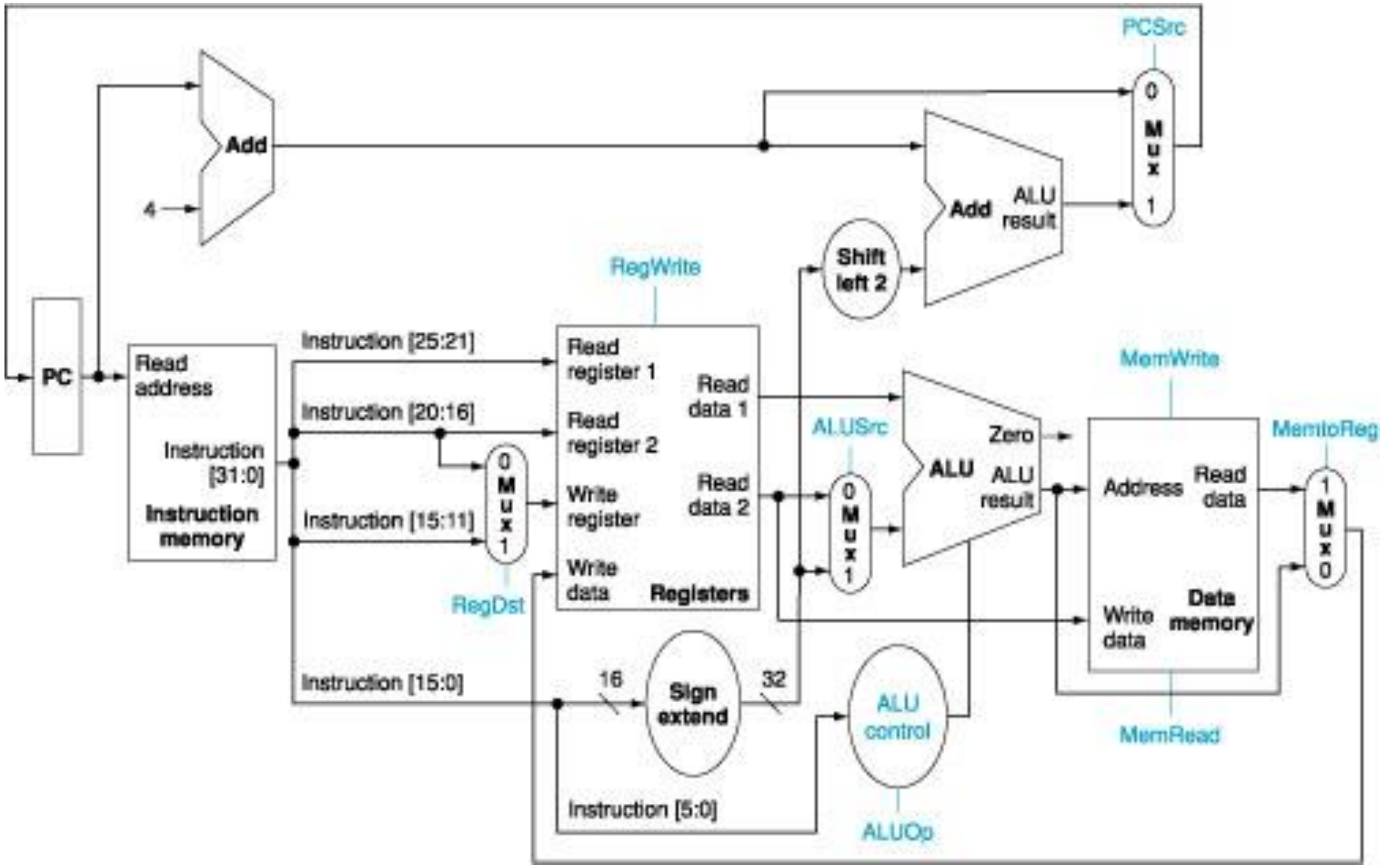
Implementing Jumps

- The one we have supports arithmetic/logic instructions, branch instructions, load and store instructions
  - We need also to support the jump instruction

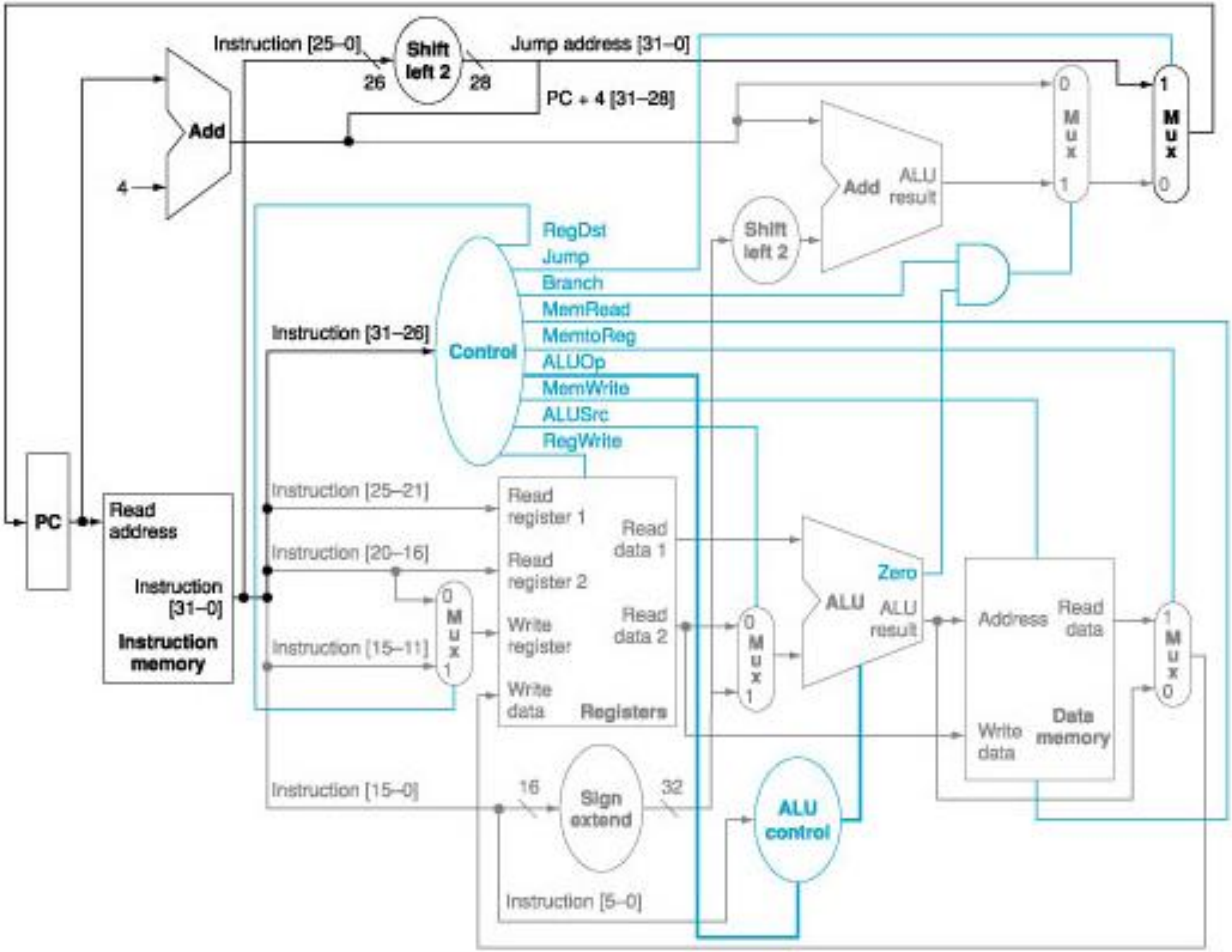


  - What are the changes we need to make?

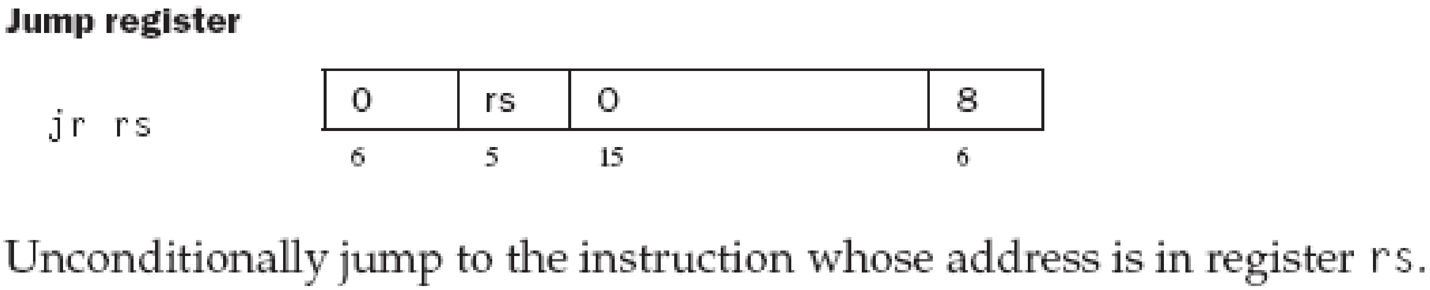
Add j?


PCSrc
Add
4
0
M
u
x
1
Add
ALU
result
Shift
left 2
RegWrite
PC
Read
address
Instruction
[31:0]
Instruction
memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
RegDst
MemWrite
ALUSrc
Zero
ALU
ALU
result
MemtoReg
Address
Read
data
Write
data
Data
memory
Instruction [15:0]
16
Sign
extend
32
ALU
control
MemRead
Instruction [5:0]
ALUOp

Supporting Jump Instruction



In Class Exercise – Supporting Jump Register



Jump register

jr rs

0	rs	0	8
6	5	15	6

Unconditionally jump to the instruction whose address is in register rs.