VE370 RC Week 5

2020/10/09

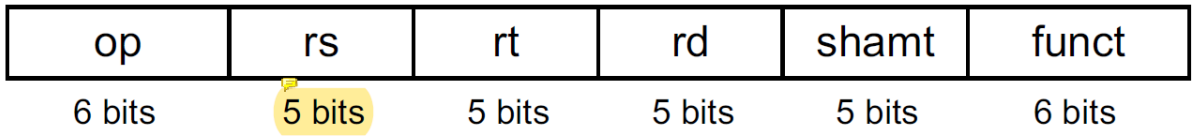
Outline

- Instruction Coding
- Digital Logic and Verilog Review

Outline

- Instruction Coding
- Digital Logic and Verilog Review

R-format



- 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)

- op, funct: check the MIPS Reference Card
- shamt: only used in shift operations

R-format Example

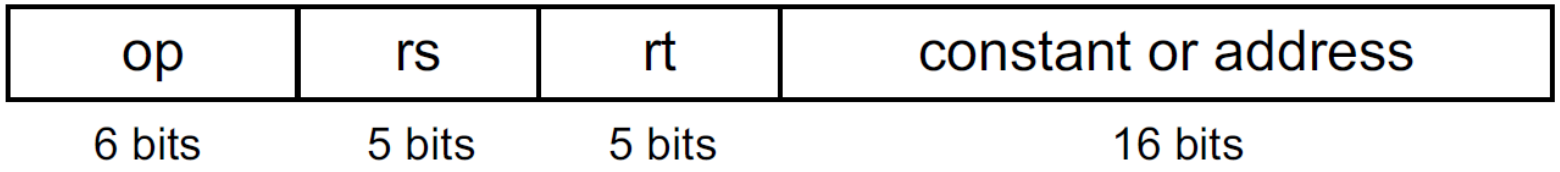
```
sll $t1, $s3, 2
```

op (6 bits)	rs (5 bits)	rt (5 bits)	rd (5 bits)	shamt (5 bits)	funct (6 bits)
0	0	$s3	$t1	2	0
0	0	19	9	2	0
000000	00000	10011	01001	00010	000000

- Result:

  $0000\ 0000\ 0001\ 0011\ 0100\ 1000\ 1000\ 0000_2 = 00134880_{16}$

I-format



16-bit immediate number or address

- rs: source or base address register
- rt: destination or source register
- Constant: -2^{15} to $+2^{15} - 1$
- Address: offset added to base address in rs

I-format Examples (T4 pp. 8 – 10)

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- `addi $t0, $s0, 4`

op	$s0	$t0	4

- `lw $t0, 4($s0)`

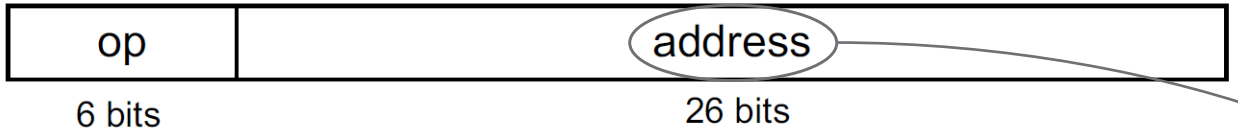
op	$s0	$t0	4

- `beq $s0, $t0, LOOP`

op	$s0	$t0	Relative Address

  - LOOP (Target address) = PC + 4 + Relative Address × 4
  - The 16-bit RA set the limitation for branch

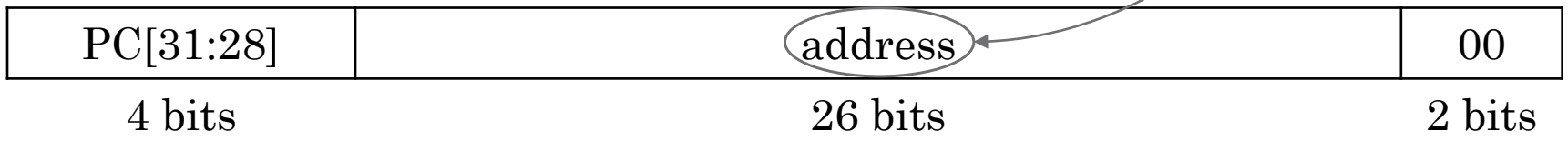
J-format



Encode full address in instruction

(Pseudo) Direct jump

- Target address = PC[31:28] : (address × 4)



- The unchanged first 4 bits set the limitation for jump
- 00 at the end: each instruction is a word

MIPS Addressing Mode (T4 pp. 15-20)

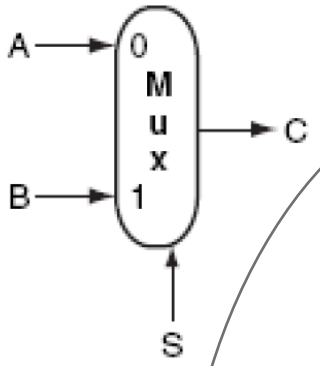
- Immediate Addressing
  - In I-type like `addi $t0, $s0, -1`
- Register Addressing
  - In R-type and I-type like `beq $s0, $s1, FUNCTION`
- **Base Addressing**
  - In I-type (memory-related) like `lw $t0, 32($s0)`
- **PC-relative addressing**
  - In near branch like `beq $s0, $s1, FUNCTION`
- **Pseudodirect addressing**
  - In J-type instructions like `j` and `jal`

Outline

- Instruction Coding
- Digital Logic and Verilog Review

2-to-1-Line Multiplexer (Mux)

- 2 ways to write 32-bit 2-to-1 Mux in Verilog




```
module mux(Out, I0, I1, Sel);
  parameter msb = 31;
  input Sel;
  input [msb:0] I0, I1;
  output [msb:0] Out;

  assign Out=(Sel==0)? I0:I1;
endmodule
```

Parameter annotation:

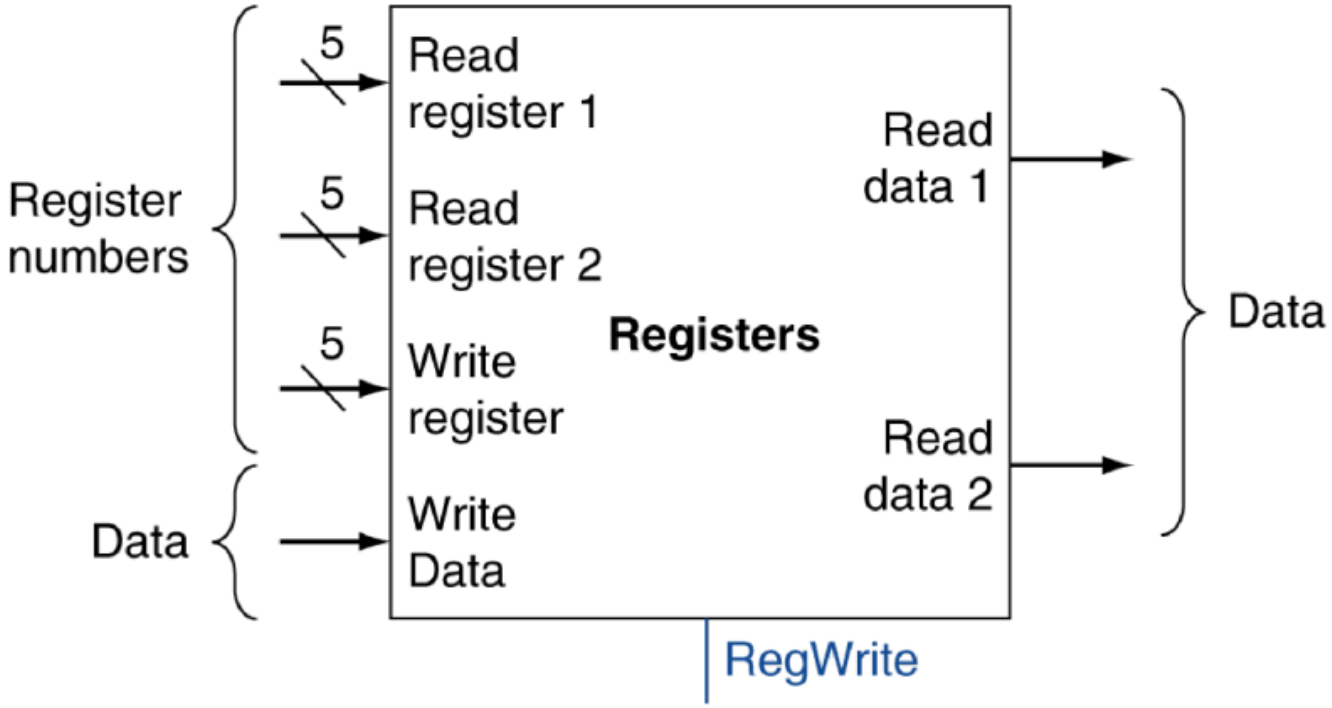
```
mux mux1(Out1, I01, I11, Sel1); // default
mux #(4) mux2(Out2, I02, I12, Sel2); //msb = 4
```

```
module mux(Out, I0, I1, Sel);
  input Sel;
  input [31:0] I0, I1;
  output reg [31:0] Out;

  always @(I0, I1, Sel) begin
    case (Sel)
       1'b0: Out = I0;
       1'b1: Out = I1;
       default Out = 0;
    endcase
  end
endmodule
```

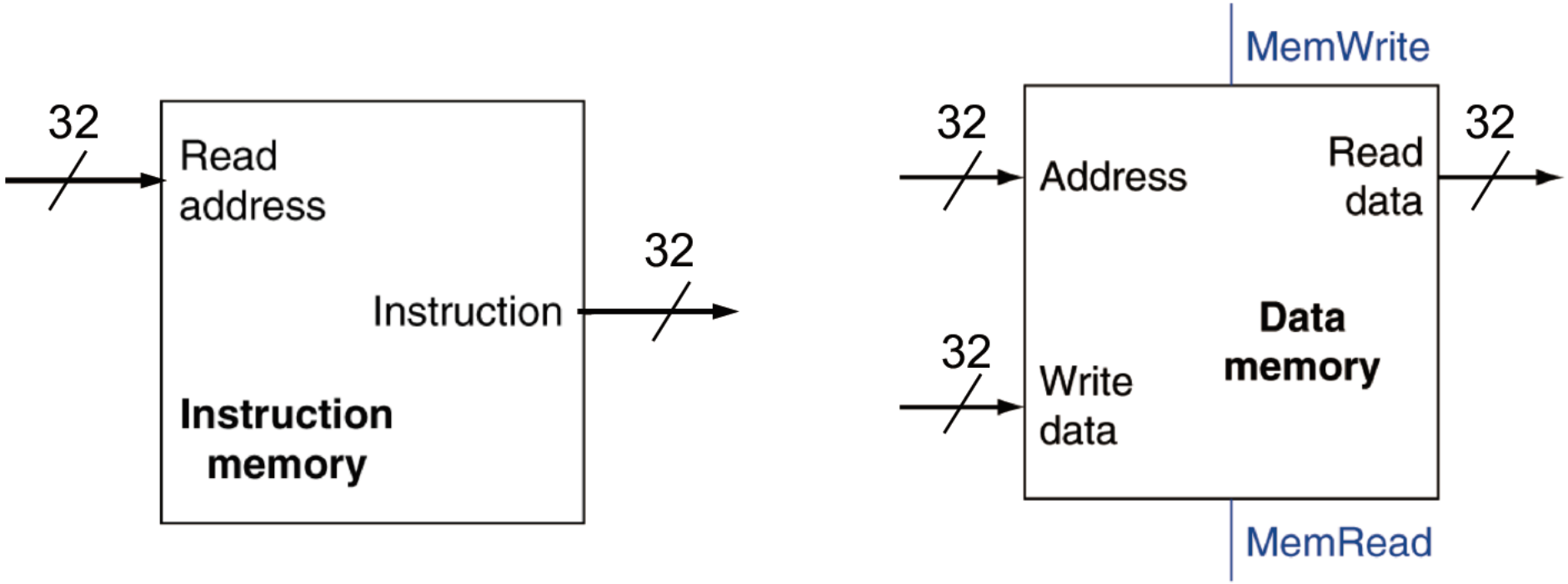
Test Bench

```
module testMux();
  reg sel;
  reg[4:0] in0,in1;
  wire[4:0] out;

  mux #(.msb(4)) mux1(out,in0,in1,sel);

  initial begin
    in0 = 5'd1;
    in1 = 5'd2;
    sel = 1'b0;
    #10 sel = 1'b1;
    #5 sel = 1'b0;
    $display("============================================");
    $display("The textual simulation results: Sel: %b, Out: %h",
sel, out);
    $display("============================================");
    #5 $finish;
  end
endmodule
```

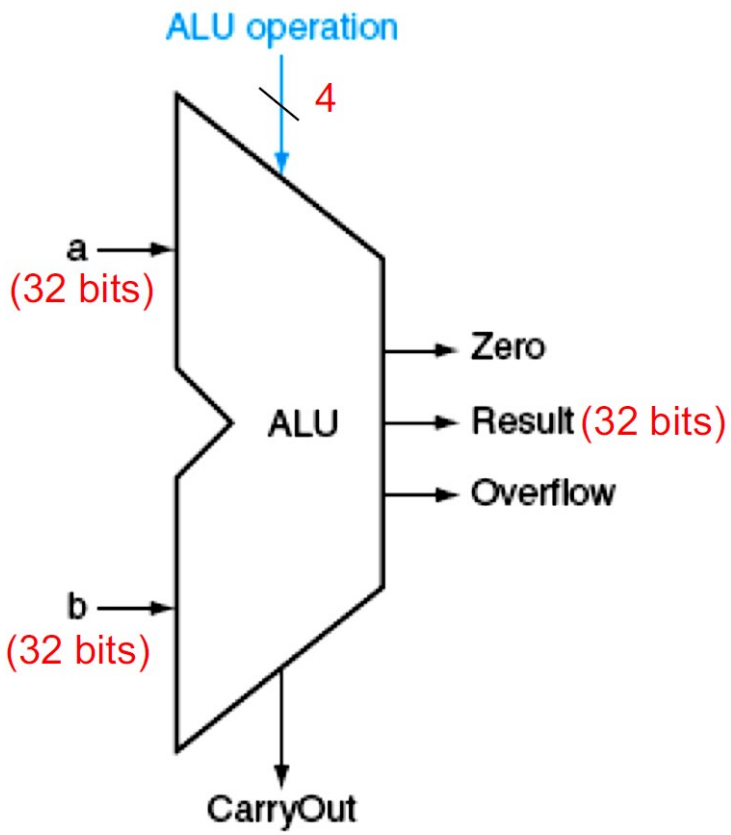
Register Files



Memory in MIPS



ALU


ALU operation
4
a
(32 bits)
b
(32 bits)
ALU
Zero
Result (32 bits)
Overflow
CarryOut

More Verilog Operations

- **`$readmemb`**`(“<file_name>”, “<memory_name>”);`
  - read text files in binary
  - could be used to load instruction memory in project 2
- **`always`** `@(*),` **`always`** `@(`**`posedge`** `clk)`
- Simulate clock in test bench: **`always`** `#delay clk = ~clk;`
- Procedural assignment
  - Left Hand Side must be reg data type
  - Blocking assignment (=), Non-blocking assignment (<=)
- Concatenation: using {}
  - e.g. `Out = {A1, A0};`
- …