Course Design of Computer Organization

Wang Zonghui

College of Computer Science & Technology,
Zhejiang University
March, 2010

topics

- 1. Verilog and Xilinx ISE
- 2. Basic logic component of datapath design
- 3. ALU and the ALU controller design
- 4. R-type instruction processor design
- 5. CPU controller design
- 6. Design of datapath of single-cycle clock CPU
- 7. Multiple-cycle clock CPU design
- 8. Microprogrammed CPU controller unit design
- 9. Design of datapath of Microprogrammed CPU

Outline

- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures

Experiment Purpose

- Understand the working principles and functions of register file, ALU and ALU Controller unit, as well as the combination methods of the components.
- Grasp the combination method of the necessary components to implement the R-type instruction’s processor functions.

Experiment Task

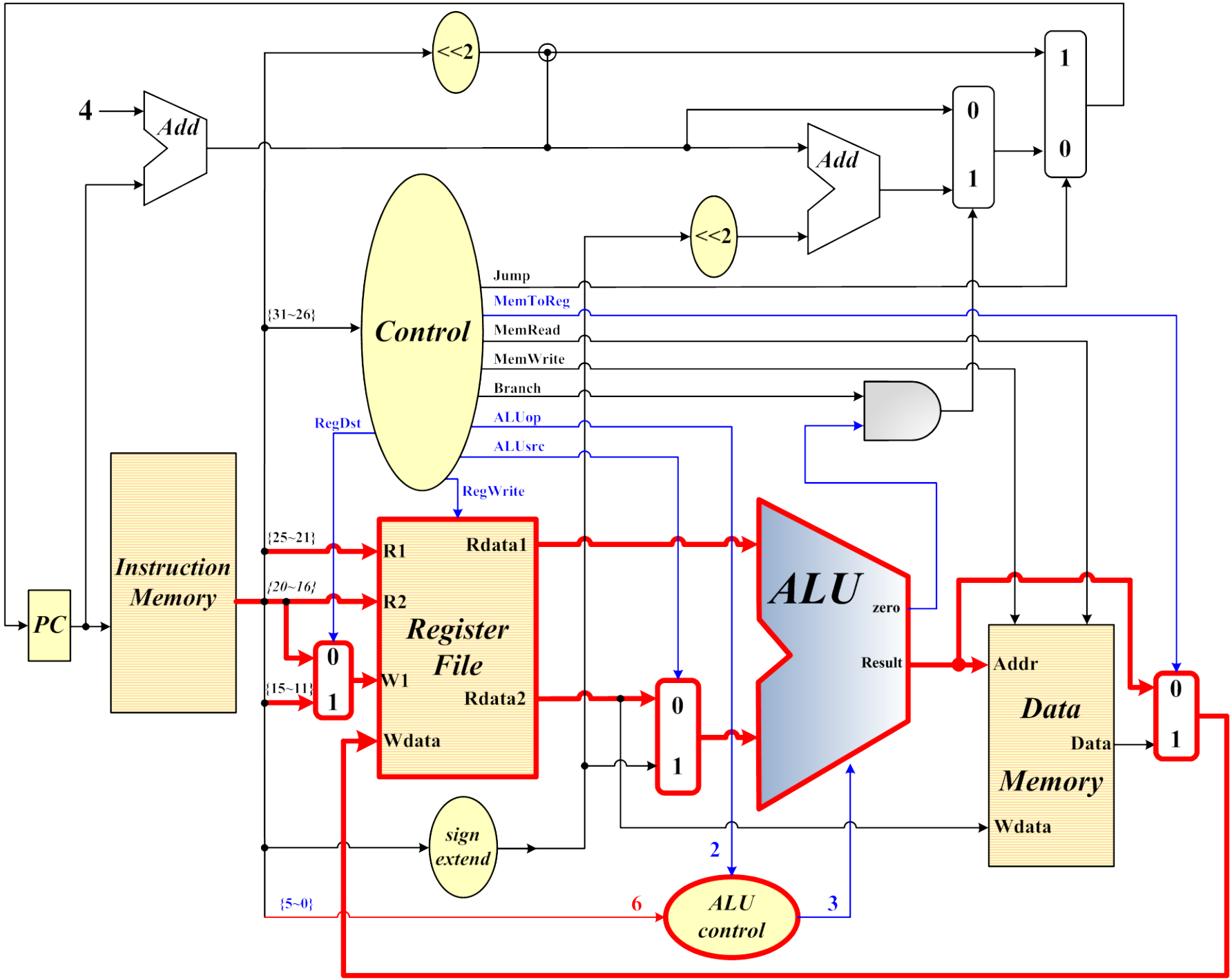
- Combine the Register File, ALU Controller, ALU and other units to implement R-type instruction processor , and simulate it.

Basic Principle

- The working procedure of R-type instruction processor.
- R-type instruction Operation

The working procedure of R-type instruction processor

1. Get the source registers' addresses.
2. Read the source registers' values from the Register File.
3. Calculate the result according to the ALU operation type.
4. Write back the result to the Register File.


4
Add
<<2
Add
<<2
0
1
1
0
Control
Jump
MemToReg
MemRead
MemWrite
Branch
ALUop
ALUsrc
RegDst
RegWrite
{31~26}
PC
Instruction Memory
{25~21}
{20~16}
{15~11}
R1
R2
W1
Wdata
Rdata1
Rdata2
Register File
0
1
ALU
zero
Result
Addr
Data
Memory
Data
Wdata
sign extend
{5~0}
2
6
3
ALU control

R-type Instr. Format

bits	3		2										1										0									
	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
R	**op = 000000: 6**						**rs: 5**						**rt: 5**					**rd: 5**					**shamt: 5**					**func: 6**				

field	signification
op	**Instruction opcode**
rs	**The first source operand register**
rt	**The second source operand register**
rd	**Dest. operand register, store the result**
shamt	**Shift amount**
funct	**Function field, to select an operation from the op field**

R-type Instructions

MIPS Instruction								
bits #	**31:26**	**25:21**	**20:16**	**15:11**	**10:6**	**5:0**	***Operation***	
R-type	**op**	**rs**	**rt**	**rd**	**sa**	**func**	**Register Operation**	**PC**
add	000000	rs	rt	rd	00000	100000	`rd = rs + rt;`	PC+=4
sub		rs	rt	rd	00000	100010	`rd = rs - rt;`	PC+=4
and		rs	rt	rd	00000	100100	`rd = rs & rt;`	PC+=4
or		rs	rt	rd	00000	100101	`rd = rs | rt;`	PC+=4
slt		rs	rt	rd	00000	101010	`if(rs < rt) rd = 1; else rd = 0;`	PC+=4
sll		00000	rt	rd	sa	000000	`rd = rt << sa;`	PC+=4
srl		00000	rt	rd	sa	000010	`rd = rt >> sa; (logical)`	PC+=4
sra		00000	rt	rd	sa	000011	`rd = rt >> sa; (arithmetic)`	PC+=4

Register File

- The 5-bit address.
- Consist of 32 32-bit registers:

Register name	Register number	Register name	Register number
$zero	0	$t8~$t9	24~25
$at	1	$k0~$k1	26~27
$v0~$v1	2~3	$gp	28
$a0~$a3	4~7	$sp	29
$t0~$t7	8~15	$s8/$fp	30
$s0~$s7	16~23	$ra	31

ALU Operation Code Table

ALUop		Func（from R instruction）						operation
ALUop1	**ALUop2**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**	
0	**0**	×	×	×	×	×	×	**010:ADD**
0	**1**	×	×	×	×	×	×	**110:SUB**
1	×	**1**	**0**	**0**	**0**	**0**	**0**	**010:ADD**
1	×	**1**	**0**	**0**	**0**	**1**	**0**	**110:SUB**
1	×	**1**	**0**	**0**	**1**	**0**	**0**	**000:AND**
1	×	**1**	**0**	**0**	**1**	**0**	**1**	**001: OR**
1	×	**1**	**0**	**1**	**0**	**1**	**0**	**111:SLT**

Experiment Content

- 1. Implement the ALU module and ALU Controller module.
- 2. Implement the Register File module , and set the initial value of register to it’s address number.
- 3. Implement the top module according to the circuit.
- 4. Simulate the top module, and analysis the waveform.

Top module

```
module Rtype(clk, rst, I, A, B, result)
  input  wire clk, rst;
  input  wire [31:0] I;
  output wire [31:0] A, B, result;
         wire [31:0] Wdat, ALUout;
         wire [2:0]  ALUoper;
  RegFile(clk, I[25:21], I[20:16],
            I[15:11], result, A, B, 1'b1);
  ALUnit(A, B, ALUoper, result, zero);
  ALUctr(2'b10, I[5:0], ALUoper);
endmodule
```

Test Instructions

bits #	31:26	25:21	20:16	15:11	10:6	5:0	*Instruction*	
R-type	op	rs	rt	rd	sa	func	*Register Operation*	*Hex Instr.*
add	000000	00101	01000	10000	00000	100000	`ADD $S0,$T5,$T0`	`32'h01A8_8020`
sub	000000	01110	01001	10001	00000	100010	`SUB $S1,$T6,$T1`	`32'h01C9_8822`
and	000000	01110	11010	10010	00000	100100	`AND $S2,$T7,$T2`	`32'h01EA_9024`
or	000000	11000	01011	10011	00000	100101	`OR  $S3,$T8,$T3`	`32'h030B_9825`
slt	000000	11001	01100	10100	00000	101010	`SLT $S4,$T9,$T4`	`32'h032C_A02A`

reg	#	reg	#	reg	#	reg	#	reg	#	reg	#
$zero	0	$a2	6	$t4	12	$s2	18	$t8	24	$s8	30
$at	1	$a3	7	$t5	13	$s3	19	$t9	25	$fp	30
$v0	2	$t0	8	$t7	14	$s4	20	$k0	26	$ra	31
$v1	3	$t1	9	$t7	15	$s5	21	$k1	27		
$a0	4	$t2	10	$s0	16	$s6	22	$gp	28		
$a1	5	$t3	11	$s1	17	$s7	23	$sp	29		

Precaution

- Simulation File: New Source->Verilog Test Fixture, then modify the initial segment.

- “clk” signal is not necessary for ALU and ALU module.