CSE 2021 COMPUTER ORGANIZATION

HUGH CHESSER
CSEB 1012U

W7-W

Agenda

Topics:

1. Register files, Decoder, Data Memory, Instruction Memory – Building Blocks
2. Complete hardware implementation of goal instructions

Patterson: Appendix C, Section 4.1, 4.2, 4.3

Reminder: Next week make-up for Labs A-D

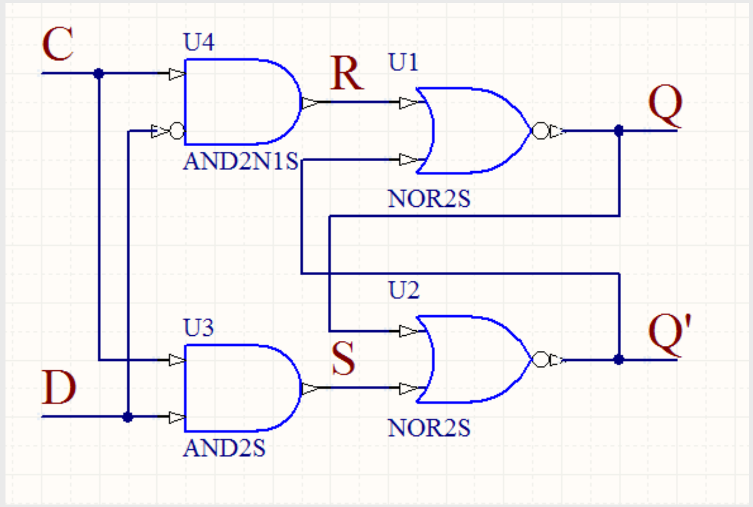
Overview (1)

Goal: Implement a subset of core instructions from the MIPS instruction set, given below

Category	Instruction	Example	Meaning	Comments
Arithmetic and Logical	**add**	`add $s1,$s2,$s3`	`$s1 ← $s2+$s3`	
	subtract	`sub $s1,$s2,$s3`	`$s1 ← $s2-$s3`	
	and	`add $s1,$s2,$s3`	`$s1 ← $s2&$s3`	**& => and**
	or	`or $s1,$s2,$s3`	`$s1 ← $s2\|$s3`	**| => or**
	slt	`slt $s1,$s2,$s3`	`If $s1 < $s3, $s1←1 else $s1←0`	
Data Transfer	**load word**	`lw $s1,100($s2)`	`$s1 ← Mem[$s2+100]`	
	store word	`sw $s1,100($s2)`	`Mem[$s2+100] ← $s1`	
Branch	**branch on equal**	`beq $s1,$s2,L`	`if($s1==$s2) go to L`	
	unconditional jump	`j 2500`	`go to 10000`	

Basics: Clocked D Latch (4)

1. For a D-latch: output Q = 1 when D = 1 (set condition)
   output Q = 0 when D = 0 (reset condition)

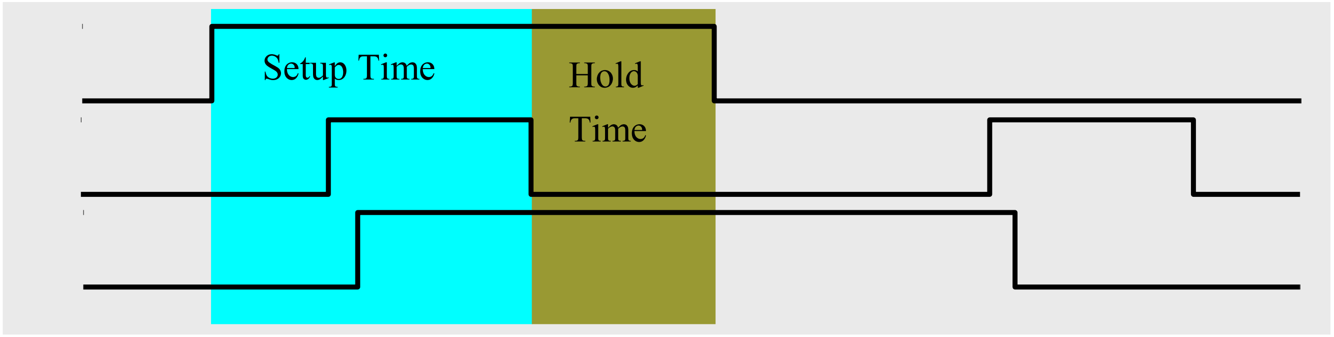


Logic Diagram

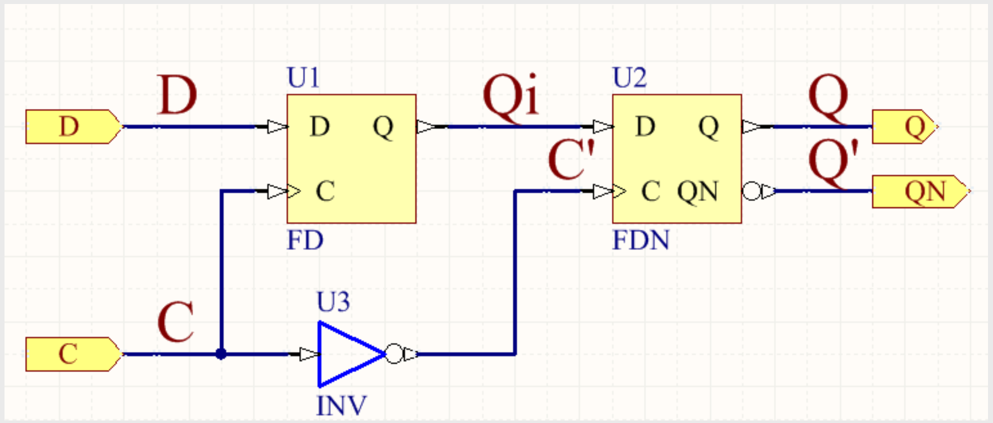
Inputs		Outputs		Comments
C	D	Q	$Q' = \bar{Q}$	
0	**X**	**Unchanged**		
1	**0**	**0**	**1**	**Reset**
1	**1**	**1**	**0**	**Set**

Function Table

2. D Latch requires clock to be asserted for output to change



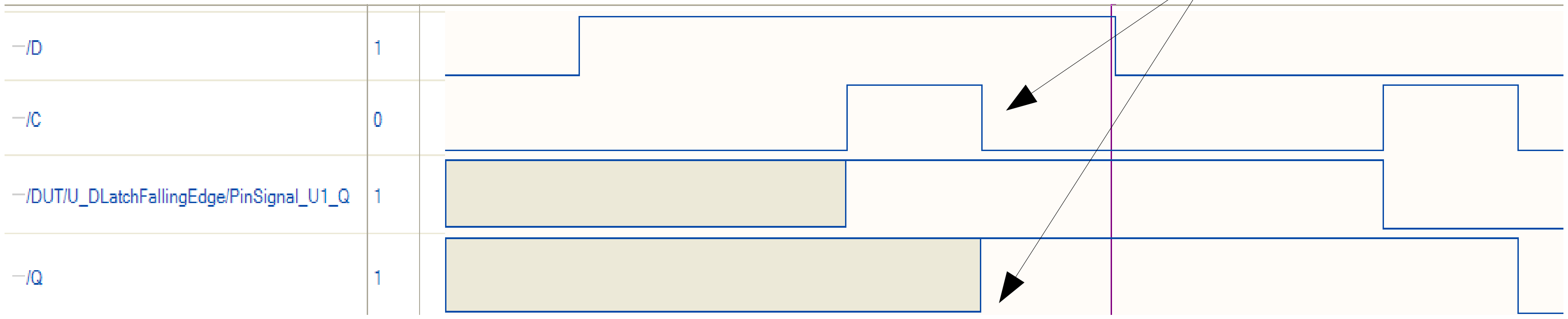
Basics: Falling Edge Triggered D flip-flop (5)



Logic Diagram

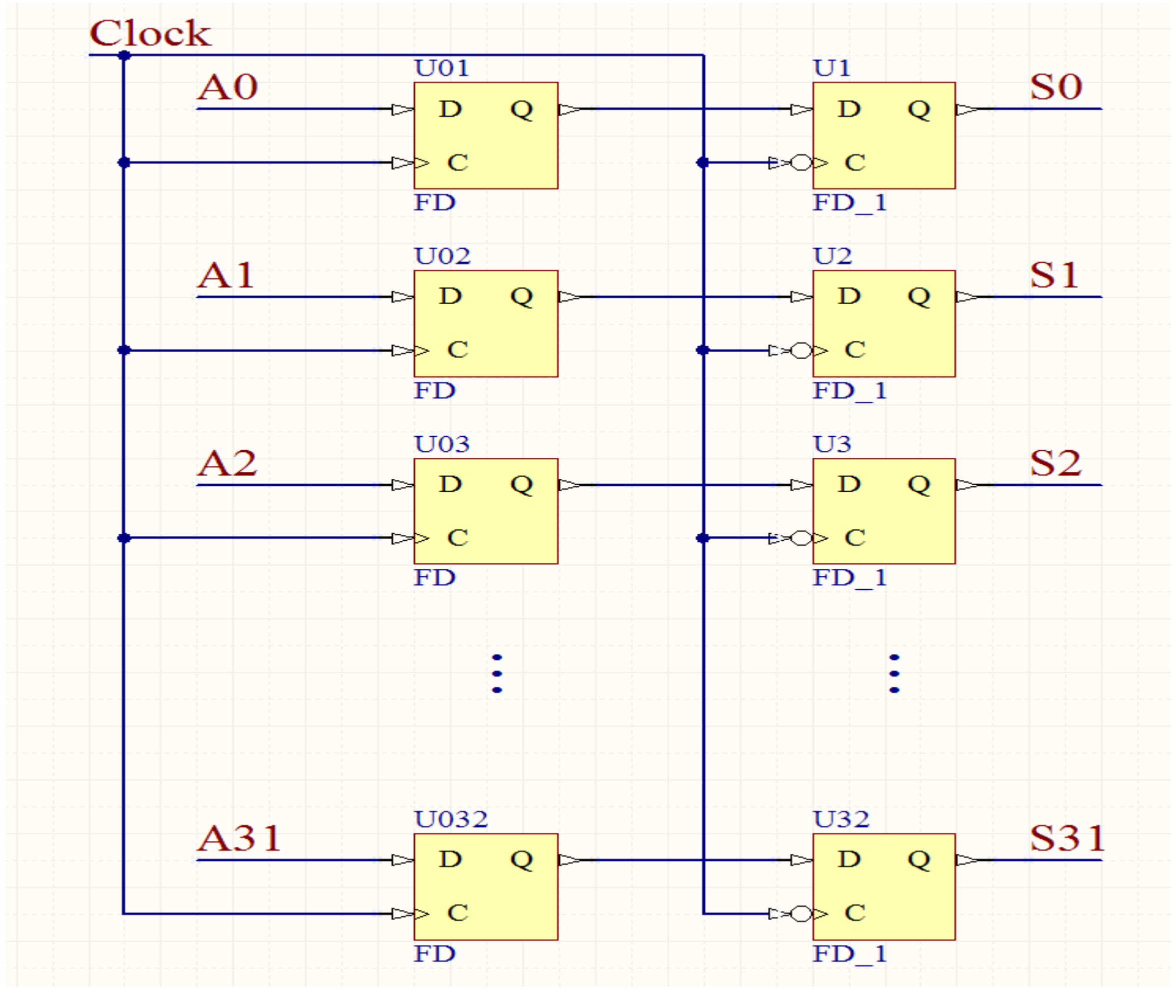
Latch changes state on falling edge of clock

Output Q follows D but changes only at the clock falling edge



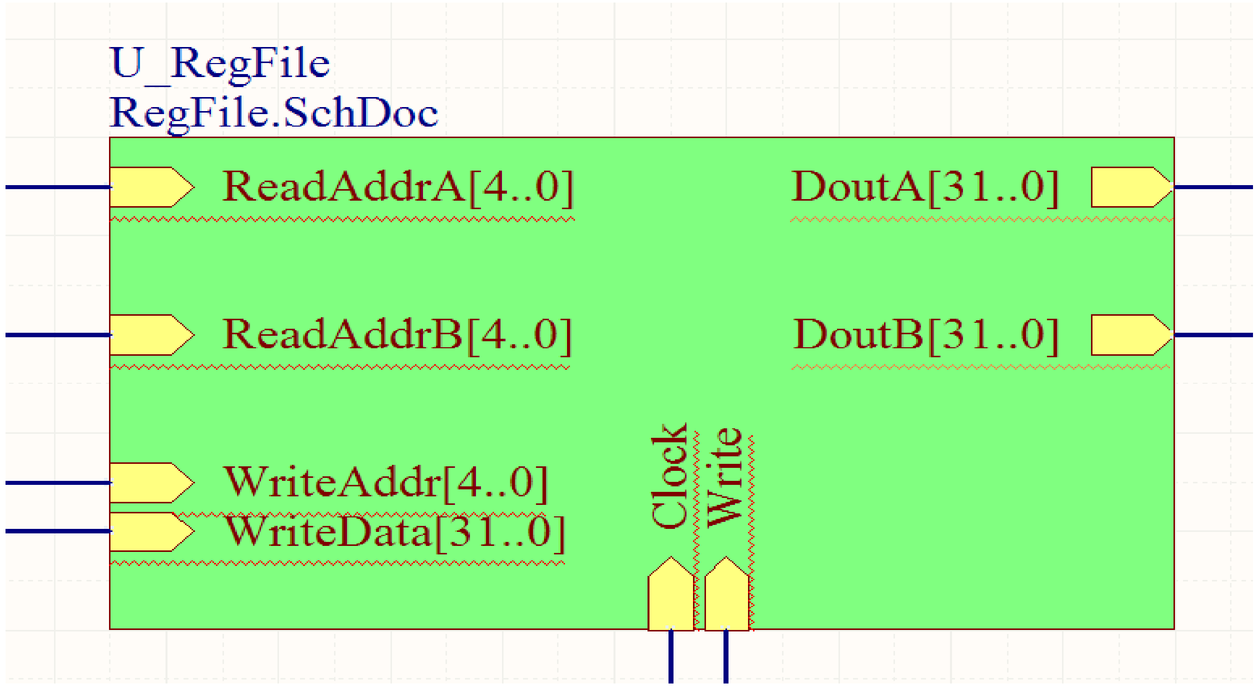
Basics: 32-bit Registers (6)

Falling edge triggered D flip-flops can be combined to form a register



Basics: Register Files (6)

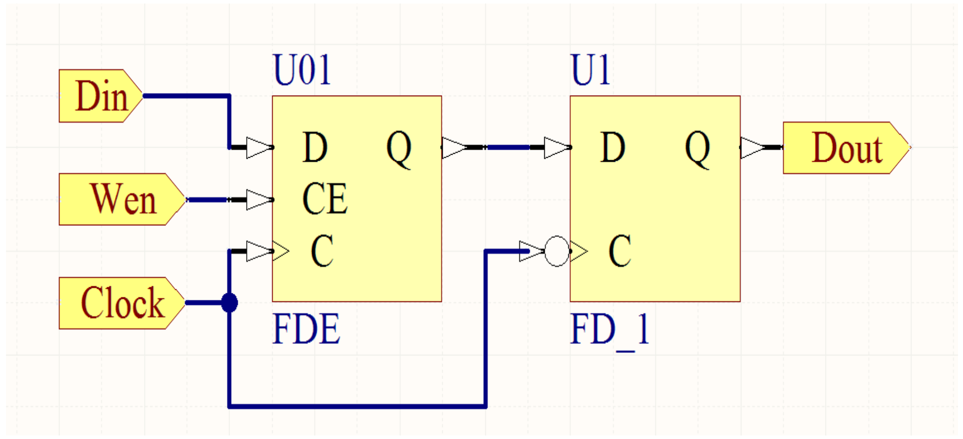
1. Register files consist of a set of registers that can be read or written individually
2. In MIPS, register file contains 32 registers
3. Two registers can be read simultaneously
4. One register can be written at one time

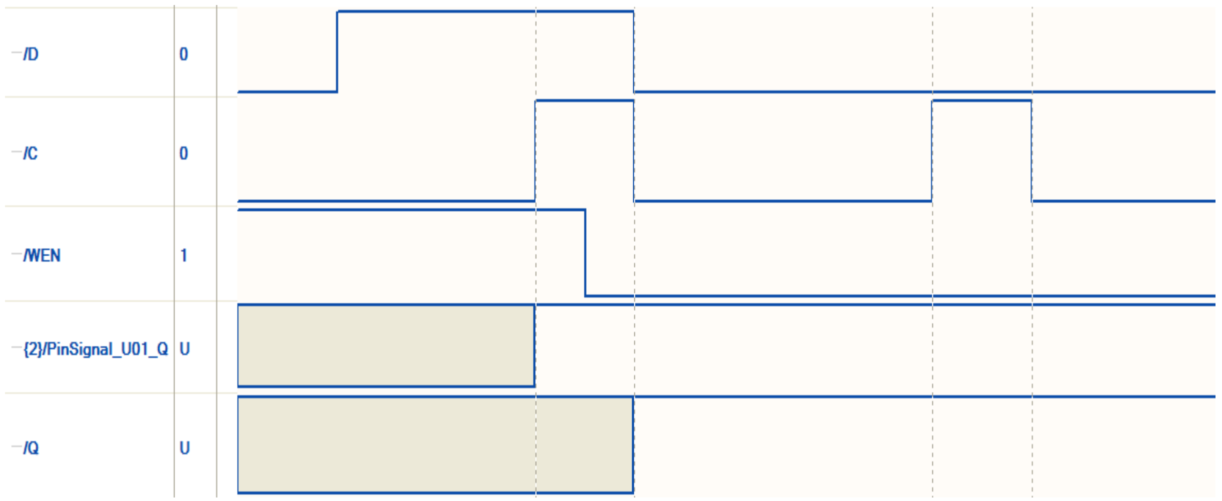


Basics: Write Enabled 1-bit Register (7)

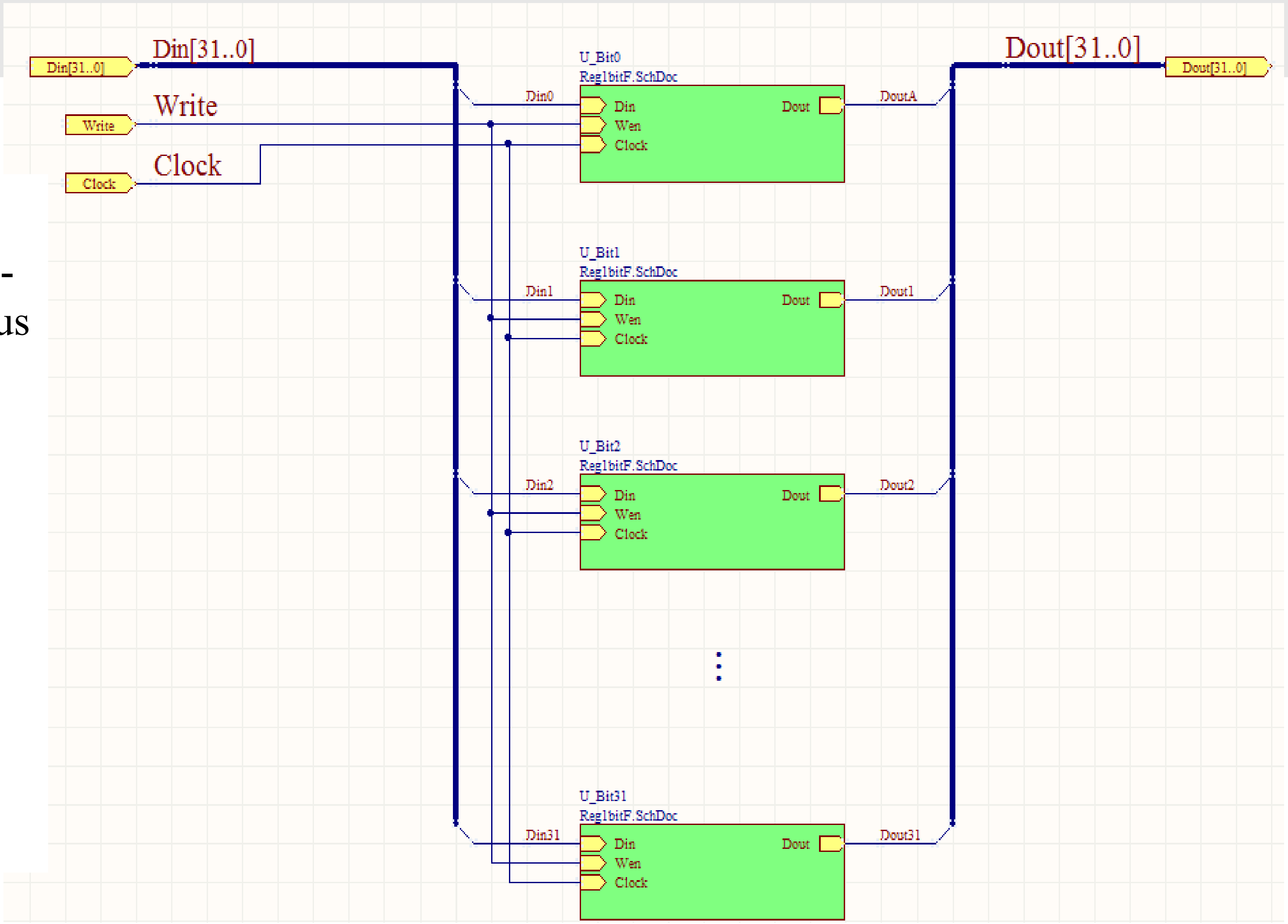
Write Operation:

- Slight change to D flip-flop to include a ”Write” input
- Din (Data input) changes flip-flop state only if “Wen” (Write enable) is true
- Clock that controls the write operation timing





Basics: 32-bit Register (8)



Register:

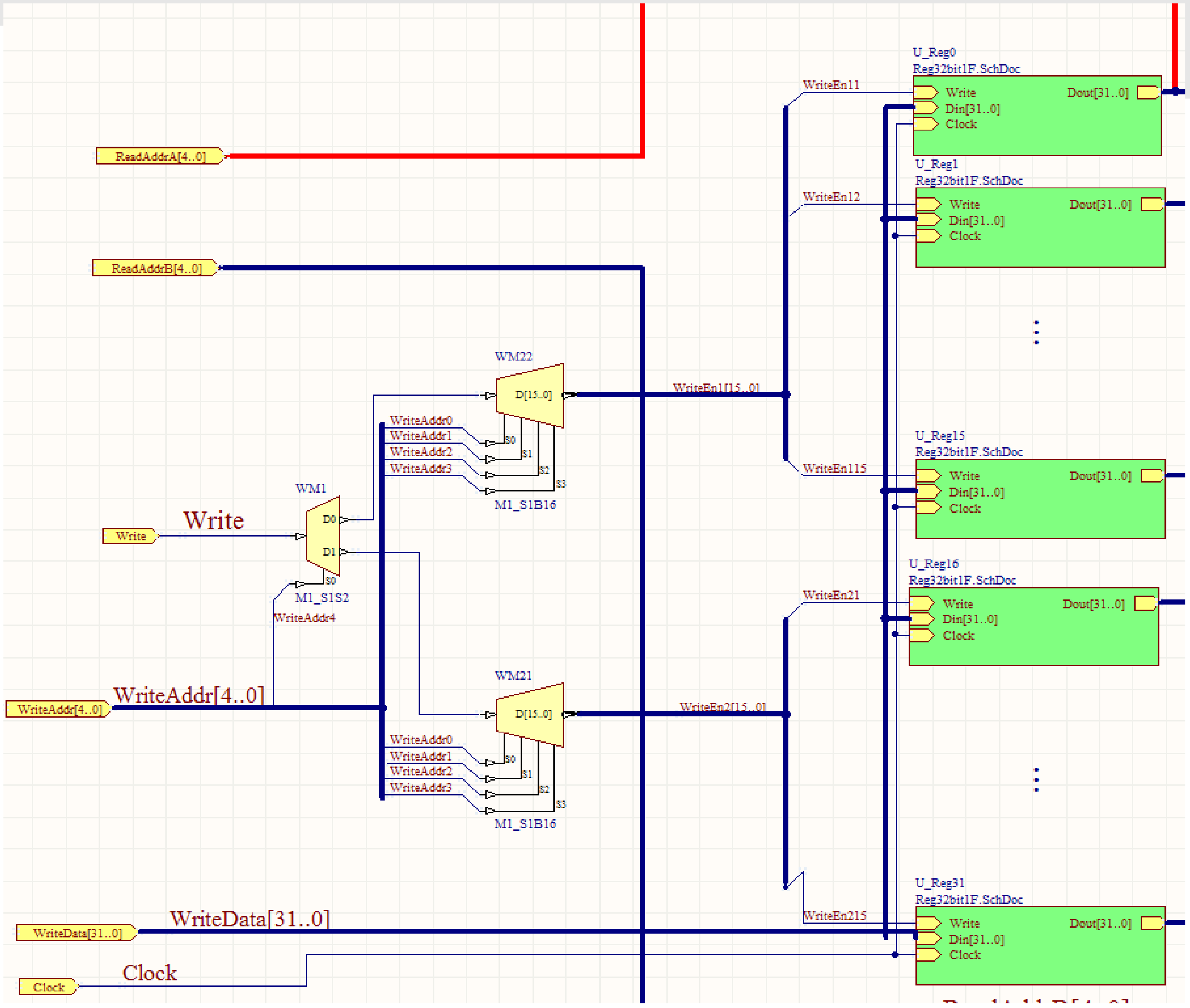
- We duplicate the Flip-flops from the previous slide to form a 32-bit register
- Each bit receives the same "Write" and Clock inputs which enable the writing of data "Din"
- A single set of Dout lines allows the register to be read

Basics: Write Operation in Register Files (9)

We now duplicate the 32-bit registers from the previous slide to provide 32 registers of the Register File

Write Operation:

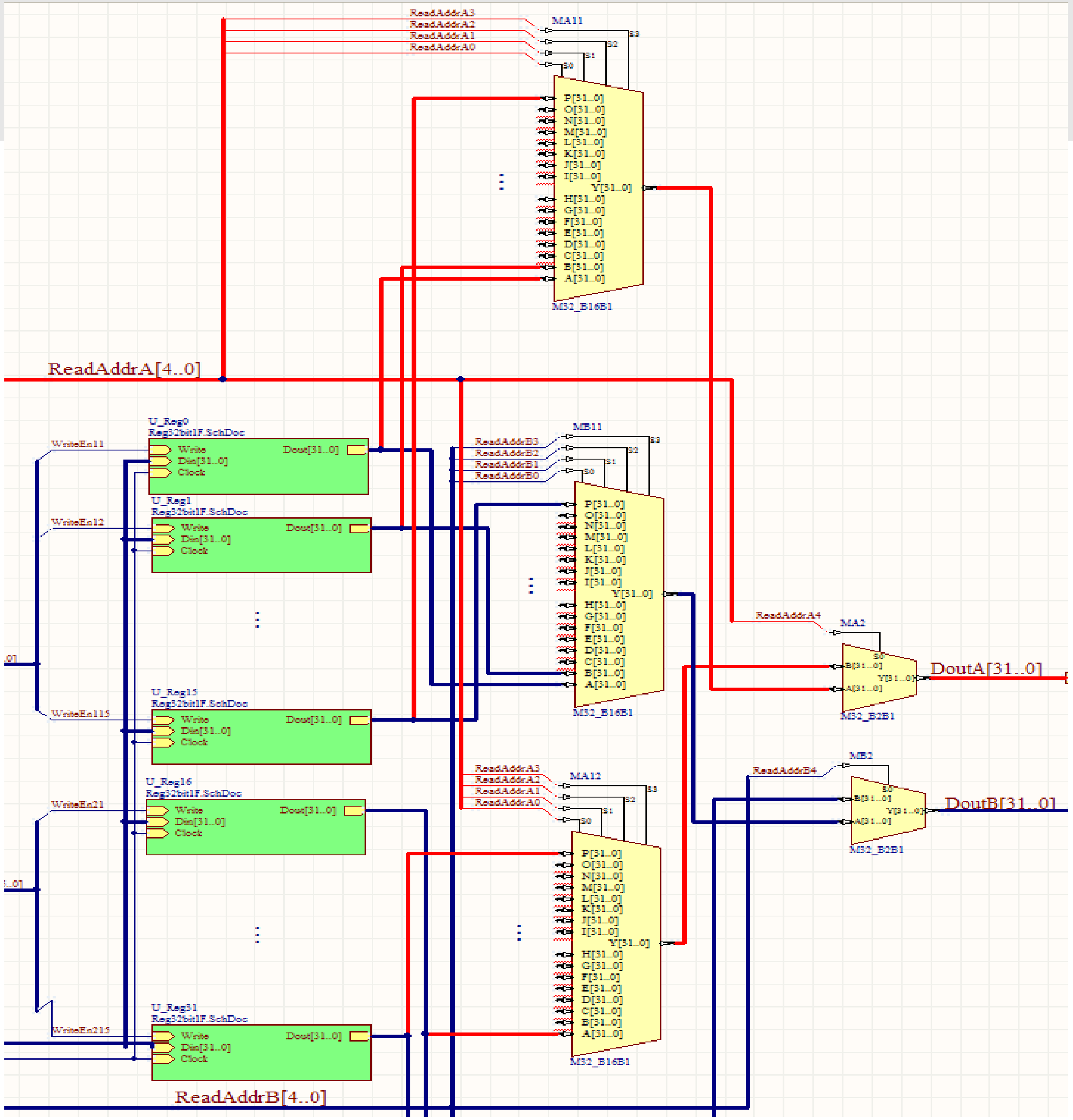
- Register number of the register to be written is one input (WriteAddr bus)
- Data to be written is the second input (WriteData bus)
- Clock that controls the write operation is the third input
- Decoders are used in the write operation



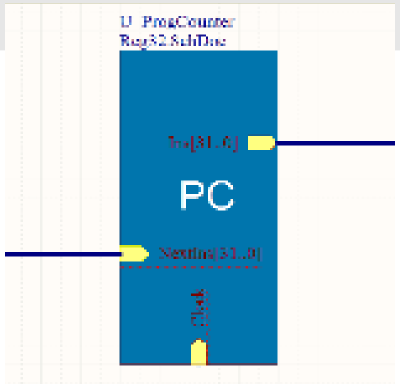
Basics: Read Operation in Register File (10)

Read Operation:

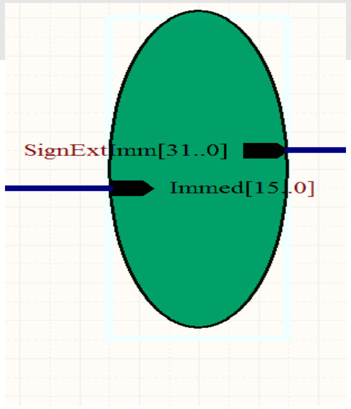
- Register number (address) of the register to be read is provided as input
- Content of the read register is the output of the register file
- Multiplexers (2 stages shown) are used in the read operation



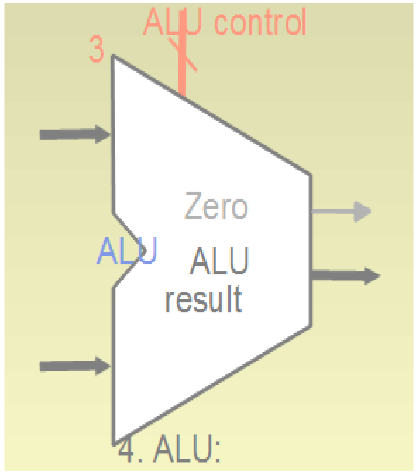
Basic Building Blocks (1)



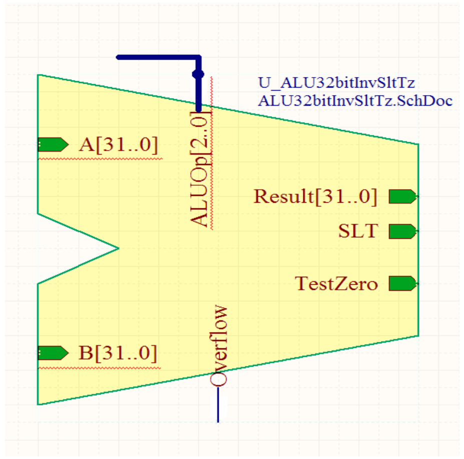
1. Program counter:
contains address of next instruction



2. Sign-extension unit:
extends a 16-bit integer to a 32-bit integer

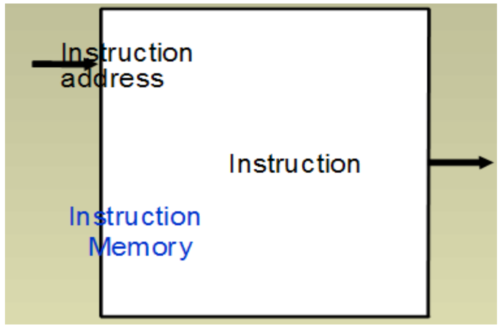


3. Adder:
adds two 32-bit integers

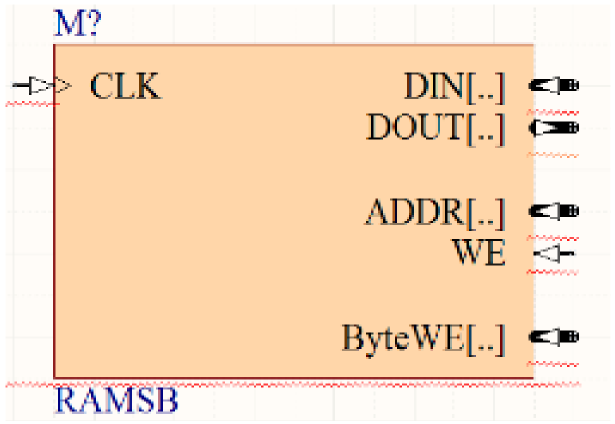


4. ALU:
add/subtract/and/or/compare two 32-bit integers

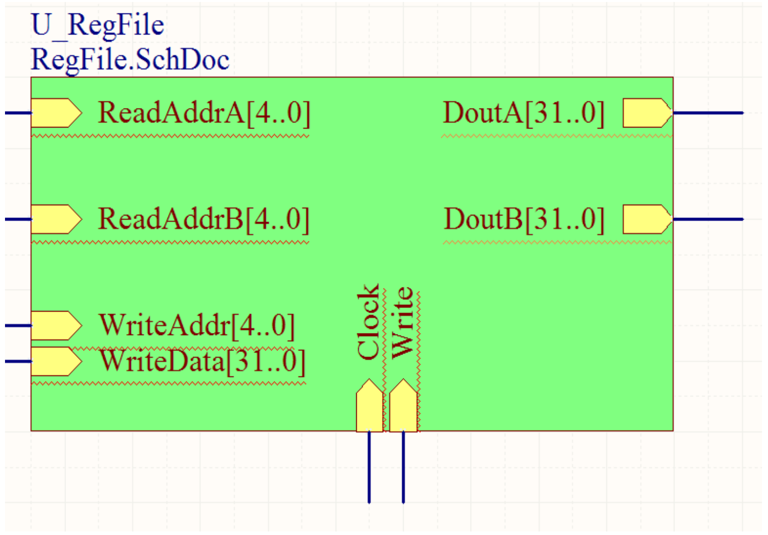
Basic Building Blocks (2)



5. Instruction memory



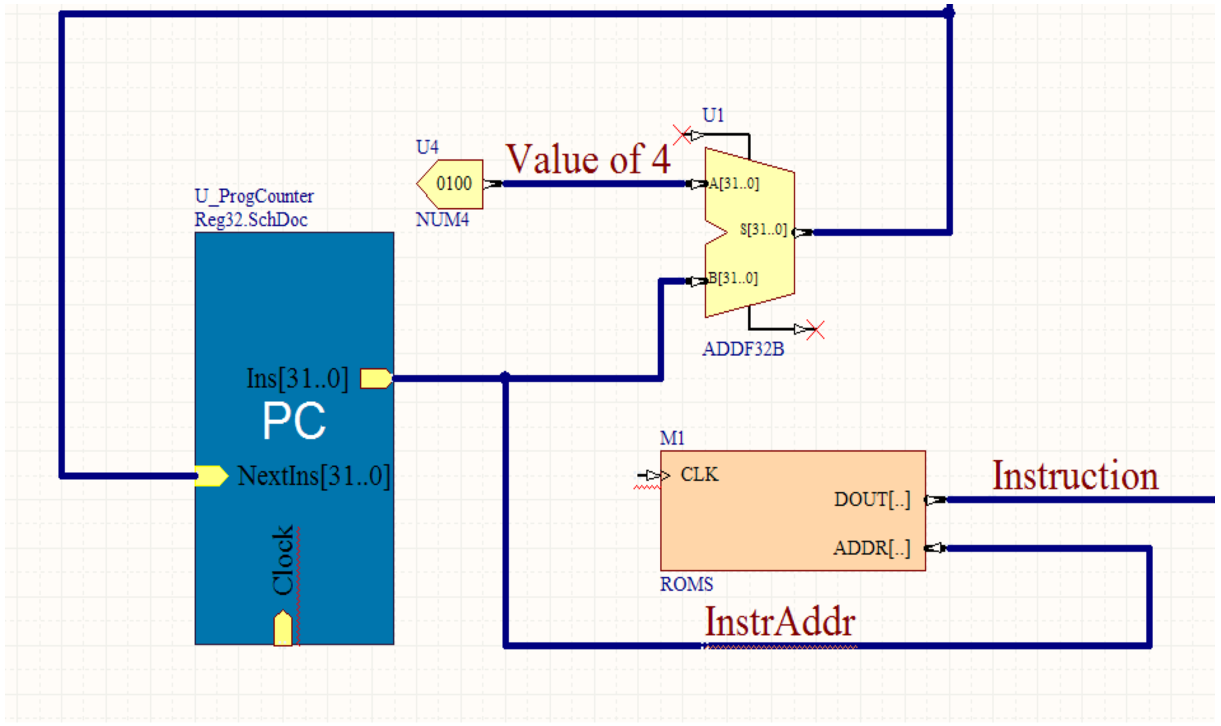
6. Data memory unit



7. Register File

Datapath: Fetch Instruction

1. Provide address from PC to Instruction Memory
2. Increment PC by 1 word (4 bytes)
3. Fetch the instruction

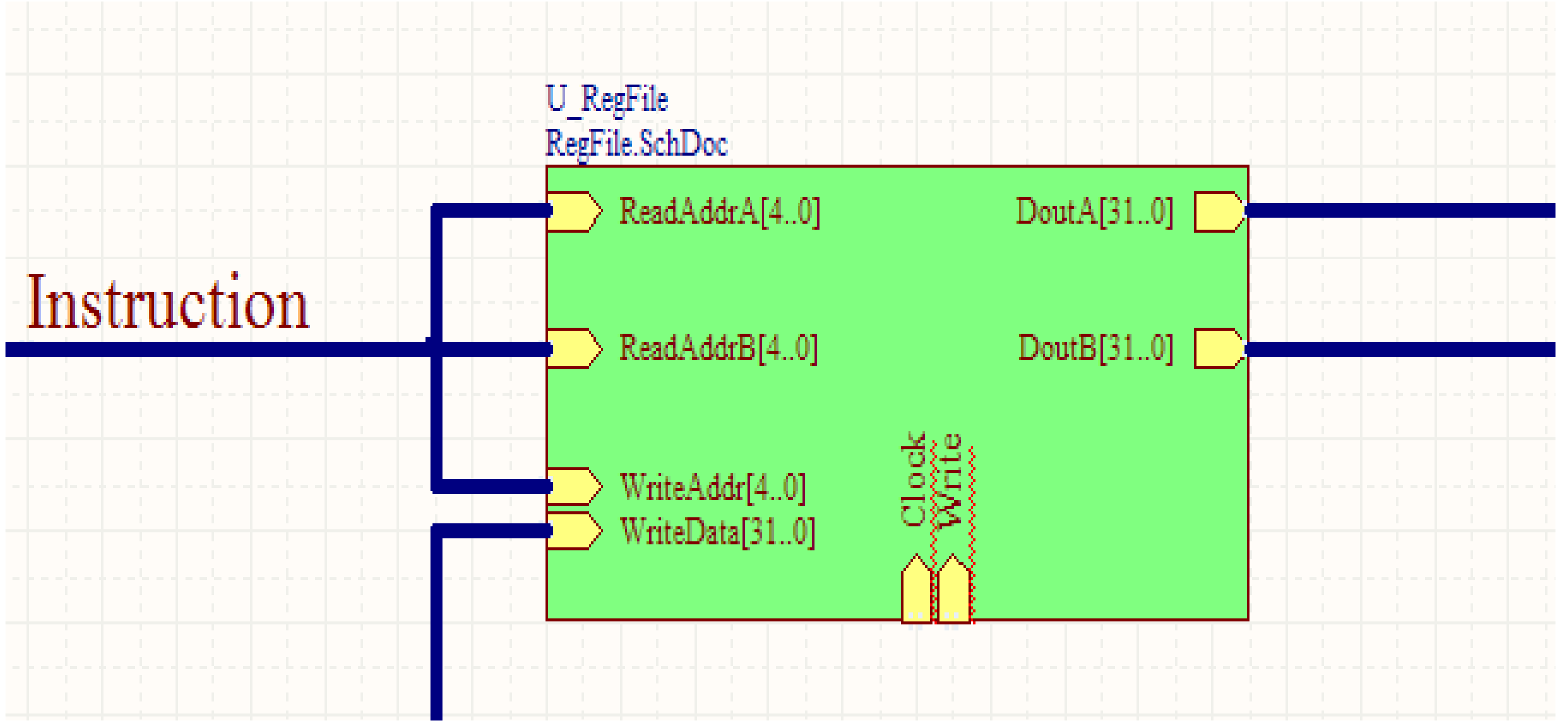


Datapath: R-type Instructions

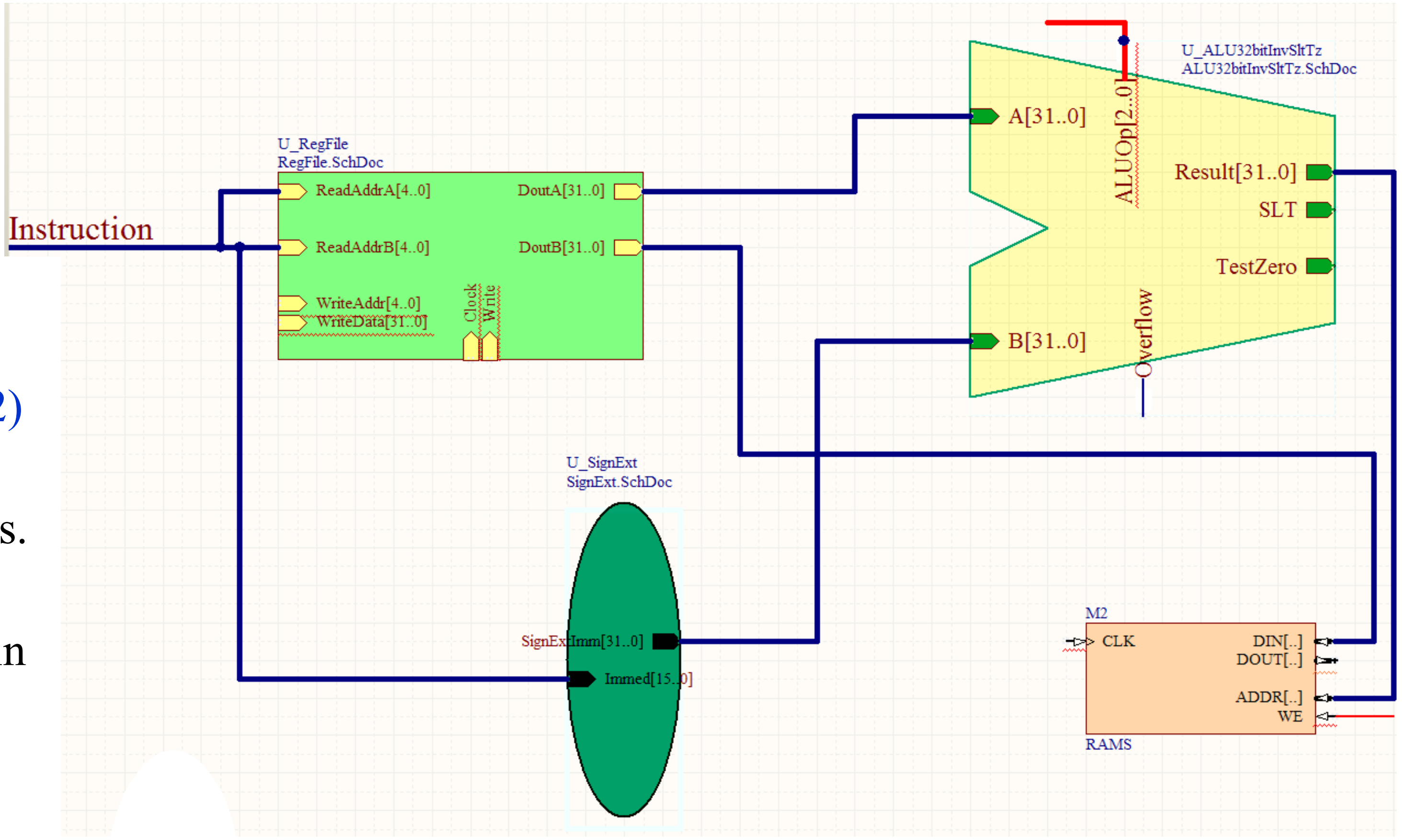
R-type instructions include arithmetic and logical instructions (add, sub, or, and, slt)

Example: `add $s1,$s2,$s3`

1. Read two registers `($s2,$s3)` specified in the instruction
2. ALU performs the required operation `(add)` on the two operands
3. Output of ALU is written to the specified register `($s1)`



Datapath: Data transfer Instruction (1)



Store instruction:

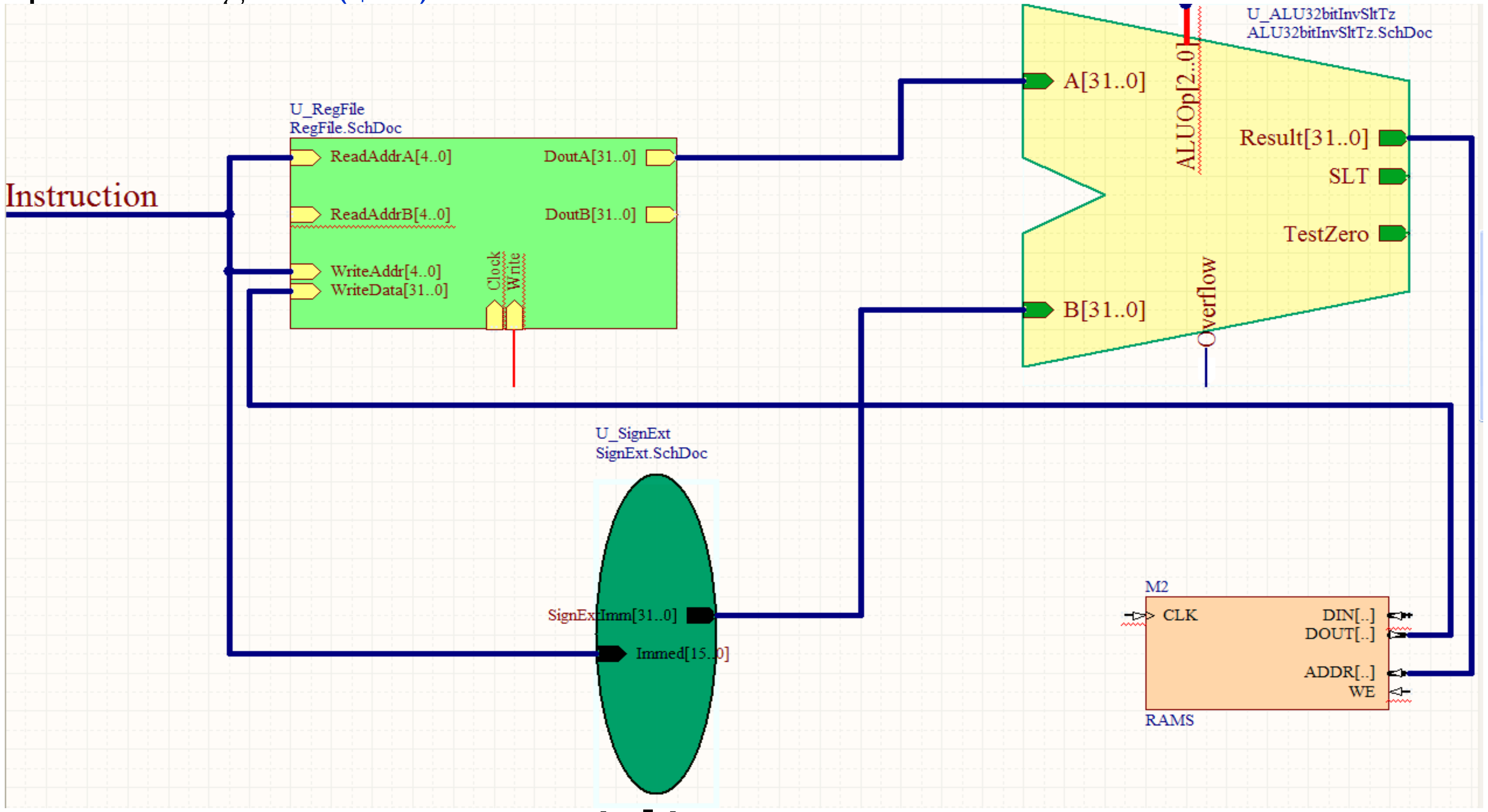
`sw $s1,offset($s2)`

1. Read two registers ($s1,$s2) specified in the instruction.
2. Offset is extended to 32 bits.
3. ALU adds offset with specified register ($s2) to obtain data memory address.
4. Address along with data of the register ($s1) to be stored passed to data memory.

Datapath: Data transfer Instruction (2)

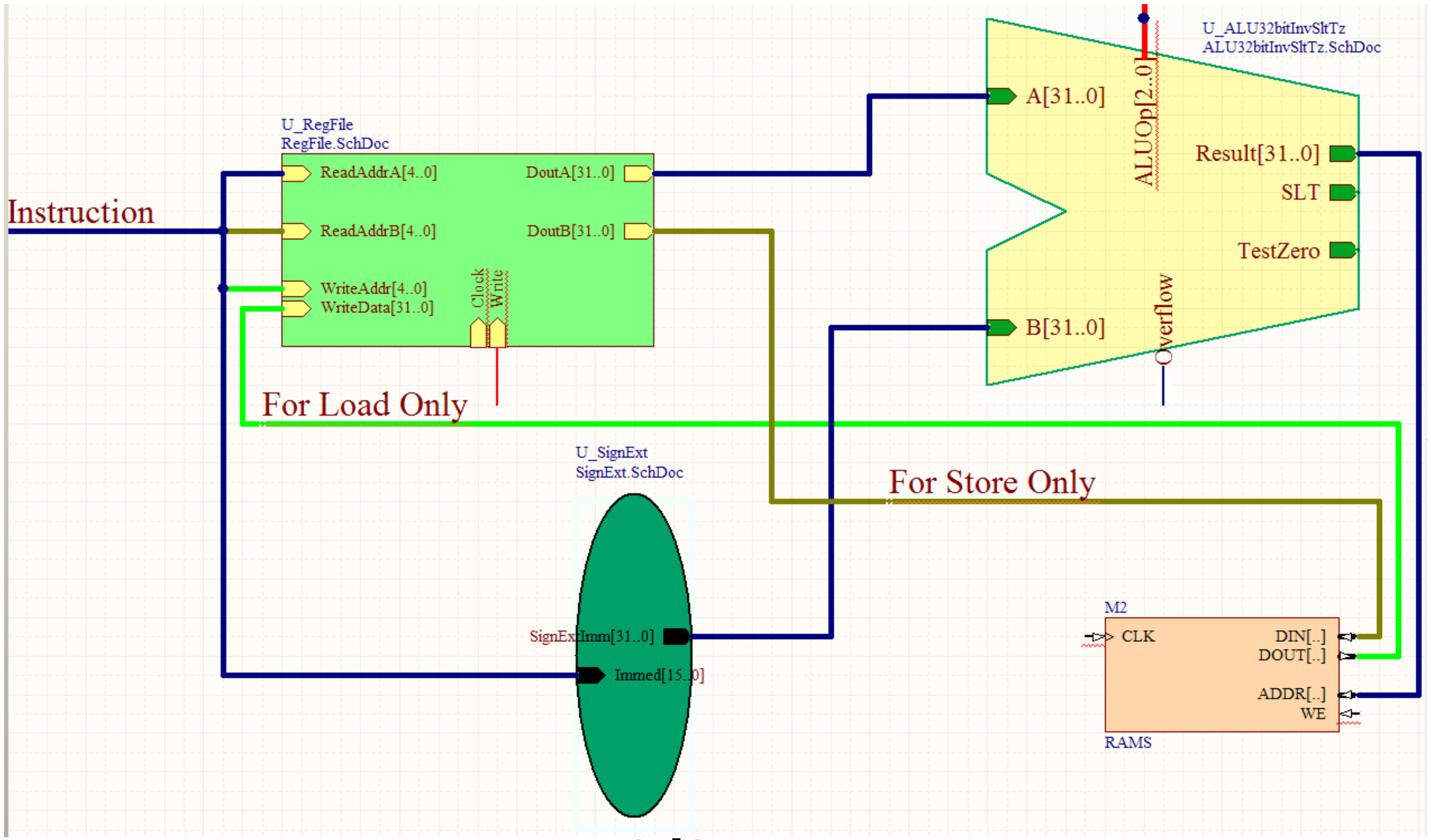
Load instruction `lw $s1,offset($s2)`

1. Read register ($s2) specified in the instruction. 2. Offset is extended to 32 bits.
3. ALU adds offset with specified register ($s2) to obtain data memory address.
4. Data memory transfers data from provided address to Register file where it is stored in the specified register ($s1).



Datapath: Data transfer Instruction (3)

Load and store instruction combined



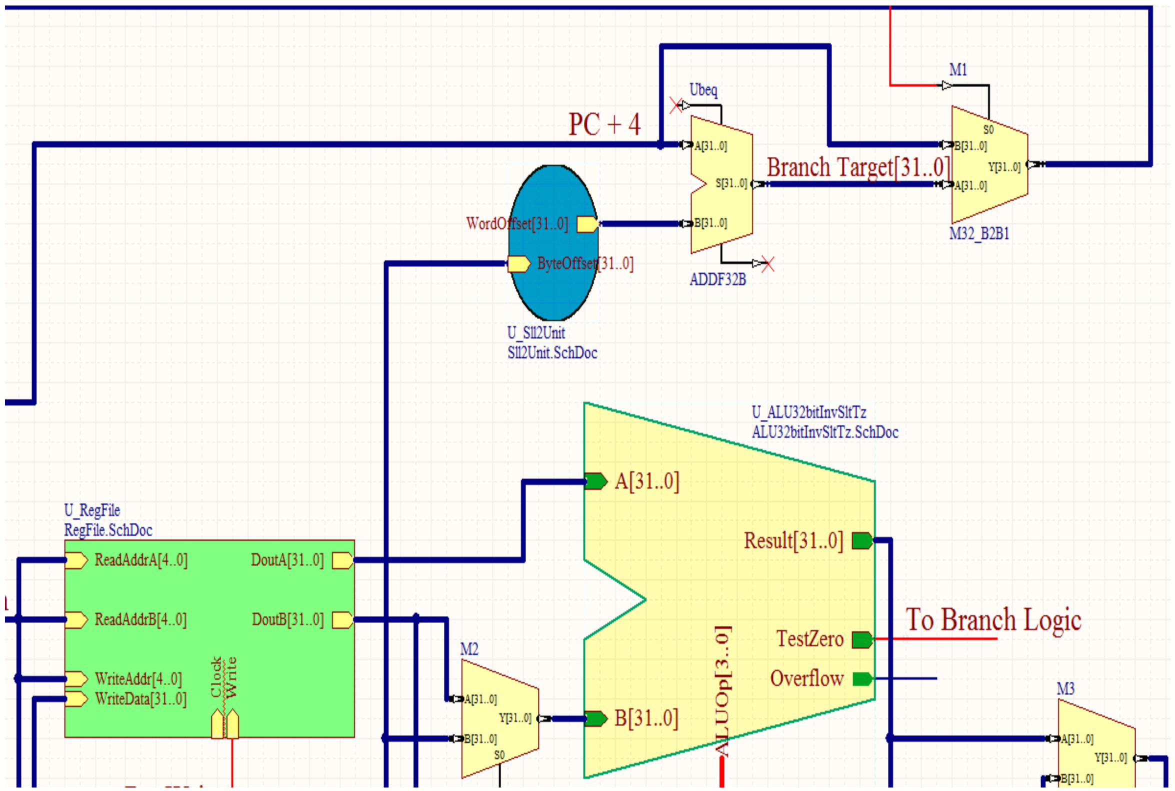
Datapath: Branch Instructions

Example: `beq $s1,$s2,Loop`

Compiler translation:

```
beq $s1,$s2,w_offset
#if $s1==$s2, goto (PC+4+4*w_offset)
```

1. Read two registers ($s2,$s3) specified in the instruction
2. ALU compares content of specified registers ($s1,$s2)
3. Adder computes the branch address
4. If equal (zero = 1), branch address is copied to PC



PC + 4
Ubeq
ADDF32B
M1
M32_B2B1
Branch Target[31..0]
WordOffset[31..0]
ByteOffset[31..0]
U_Sll2Unit
Sll2Unit.SchDoc
U_ALU32bitInvSltTz
ALU32bitInvSltTz.SchDoc
A[31..0]
B[31..0]
Result[31..0]
TestZero
Overflow
ALUOp[3..0]
To Branch Logic
U_RegFile
RegFile.SchDoc
ReadAddrA[4..0]
ReadAddrB[4..0]
WriteAddr[4..0]
WriteData[31..0]
DoutA[31..0]
DoutB[31..0]
Clock
Write
M2
M3