LecLabMeet-12-MIPS-vhdl

a walkthrough
part-1-and-2

Sachin B. Patkar

EE-705-spring-sem-2020-21 VLSI Design Lab
Mon-22-Feb-2021

Based on

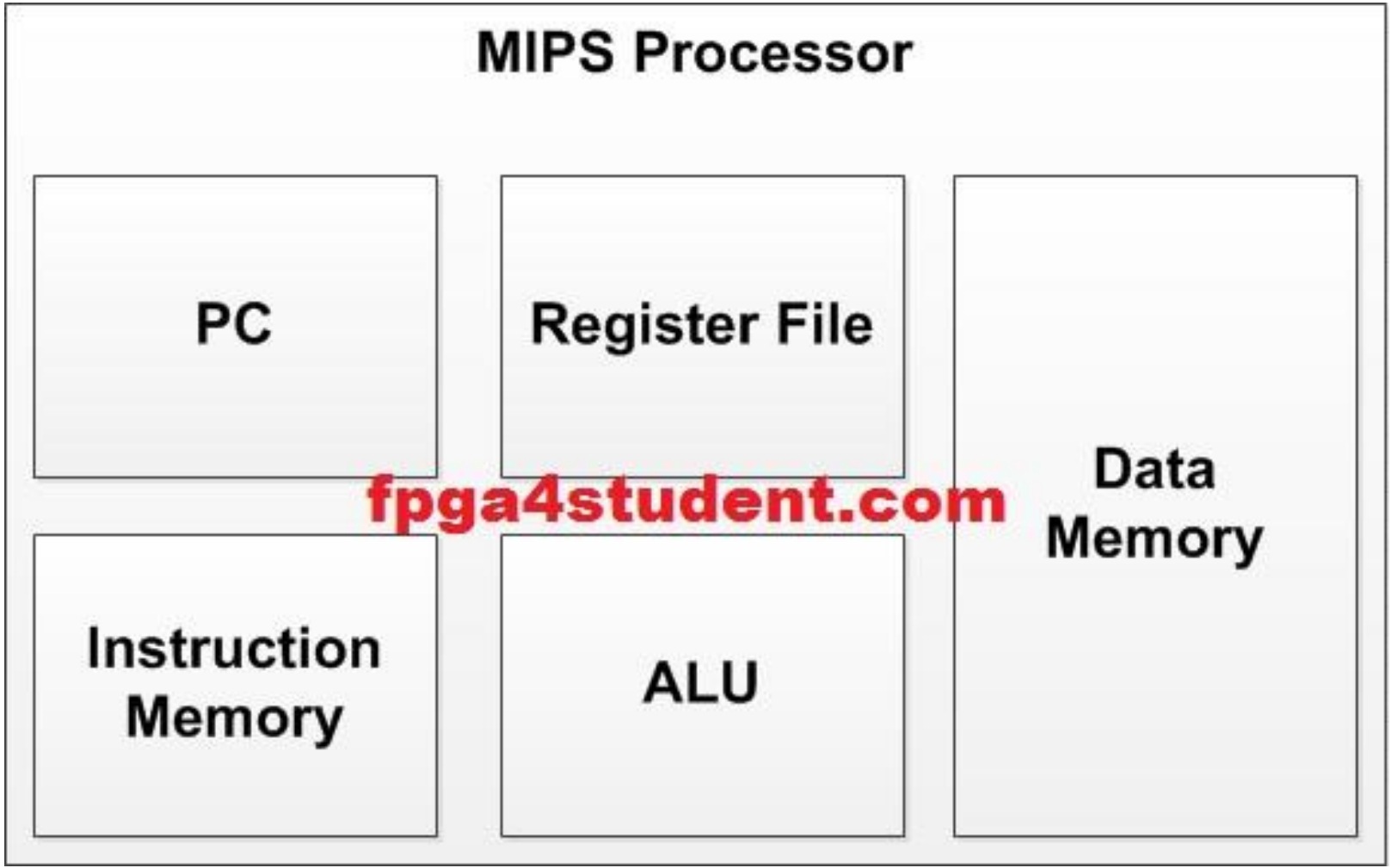
https://www.fpga4student.com/2017/09/vhdl-code-for-mips-processor.html

ISA, uArch etc

The Instruction Format and Instruction Set Architecture for the 16-bit single-cycle MIPS are as follows:

Name	Fields					Comments
Field size	3 bits	3 bits	3 bits	3 bits	4 bits	All MIPS-L instructions 16 bits
R-format	op	rs	rt	rd	funct	Arithmetic instruction format
I-format	op	rs	rt	Address/immediate		Transfer, branch, immediate format
J-format	op	target address				Jump instruction format

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	0	2	3	1	0	add $1,$2,$3
sub	R	0	2	3	1	1	sub $1,$2,$3
and	R	0	2	3	1	2	and $1,$2,$3
or	R	0	2	3	1	3	or $1,$2,$3
slt	R	0	2	3	1	4	slt $1,$2,$3
jr	R	0	7	0	0	8	jr $7
lw	I	4	2	1	7		lw $1, 7 ($2)
sw	I	5	2	1	7		sw $1, 7 ($2)
beq	I	6	1	2	7		beq $1,$2, 7
addi	I	7	2	1	7		addi $1,$2, 7
j	J	2	500				j 1000
jal	J	3	500				jal 1000
slti	I	1	2	1	7		slti $1,$2, 7


MIPS Processor
PC
Register File
Data Memory
fpga4student.com
Instruction Memory
ALU

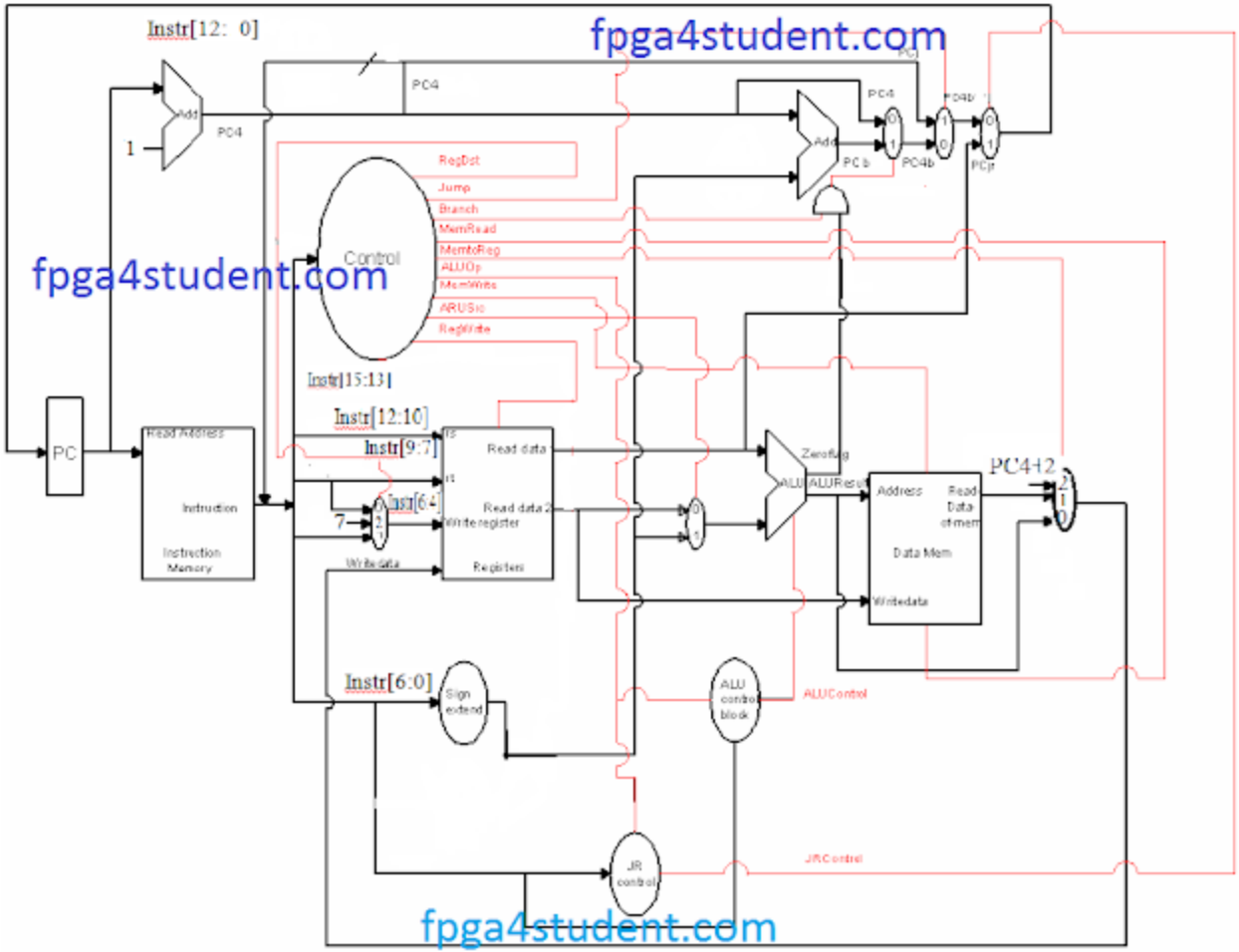
1. **Add : R[rd] = R[rs] + R[rt]**
2. **Subtract : R[rd] = R[rs] - R[rt]**
3. **And: R[rd] = R[rs] & R[rt]**
4. **Or : R[rd] = R[rs] | R[rt]**
5. **SLT: R[rd] = 1 if R[rs] < R[rt] else 0**
6. **Jr: PC=R[rs]**
7. **Lw: R[rt] = M[R[rs]+SignExtImm]**
8. **Sw : M[R[rs]+SignExtImm] = R[rt]**
9. **Beq : if(R[rs]==R[rt]) PC=PC+1+BranchAddr**
10. **Addi: R[rt] = R[rs] + SignExtImm**
11. **J : PC=JumpAddr**
12. **Jal : R[7]=PC+2;PC=JumpAddr**
13. **SLTI: R[rt] = 1 if R[rs] < imm else 0**

SignExtImm = { 9{immediate[6]}, imm

JumpAddr = { (PC+1)[15:13], address}

BranchAddr = { 7{immediate[6]}, immediate, 1'b0 }

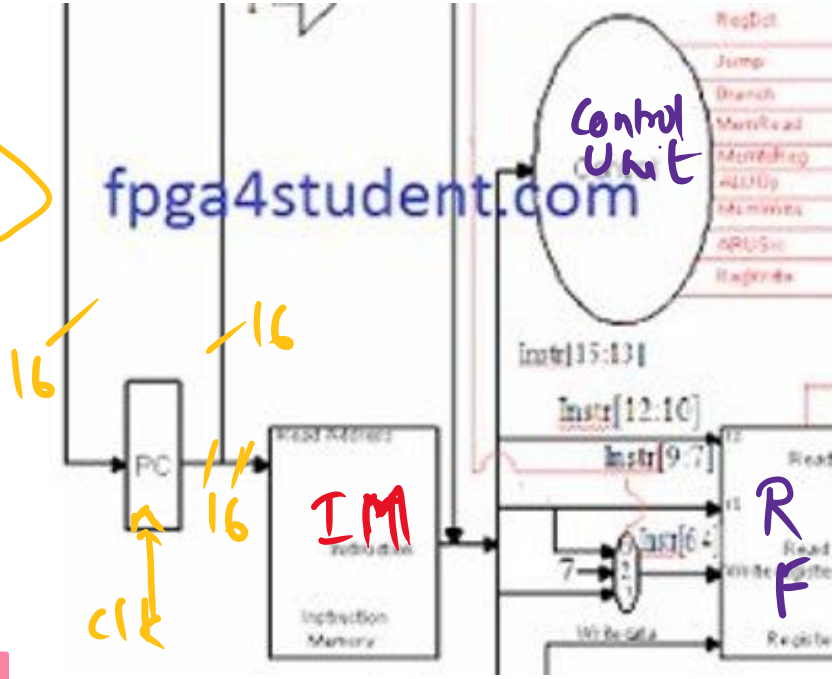
Caution : A couple of minor discrepancies between the (register-transfer-notation) RTN description of Instructions and their actual implementation in fpga4student's vhdl code


Instr[12: 0]
fpga4student.com
PC4
Add
1
PC4
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
fpga4student.com
Instr[15:13]
Instr[12:10]
Instr[9:7]
Instr[6:4]
7
PC
Instruction
Instrection Memory
Read data
Read data 2
Write register
Registers
Write data
Zero flag
ALU
ALU Result
Address
Read Data of mem
Data Mem
Write data
PC4+2
Instr[6:0]
Sign extend
ALU control block
ALUControl
JR control
JRControl
fpga4student.com

Let's correlate
HDL code fragments
with
parts of Microarchitecture !

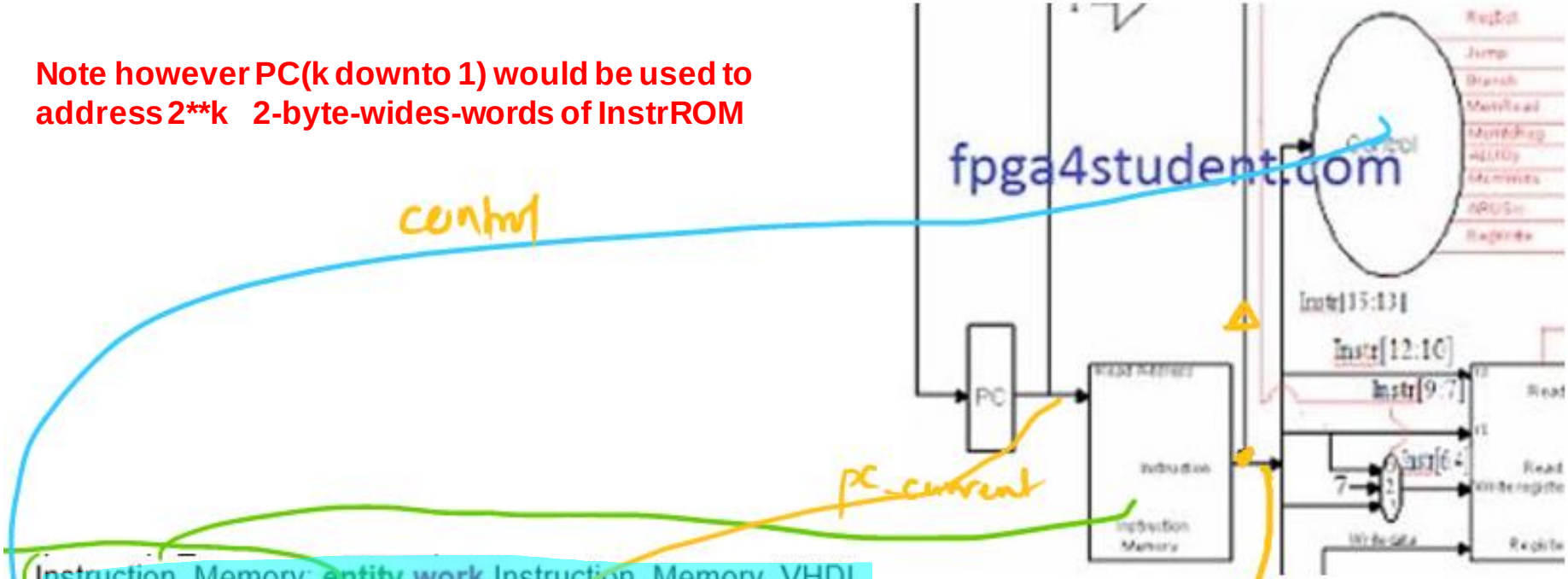
```
-- PC of the MIPS Processor in VHDL
process(clk,reset) begin
   if (reset='1') then pc_current <= x"0000";
   elsif (rising_edge(clk)) then pc_current <= pc_next;
   end if;
end process;

-- PC + 2
pc2 <= pc_current + x"0002";
```

-- PC is assumed to contain byte-address, -
- and not word address, hence +2



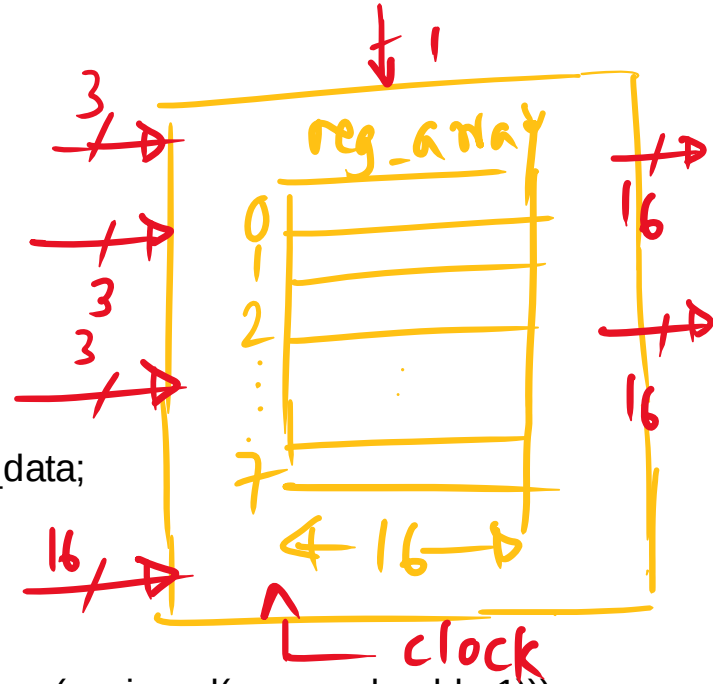

```
Instruction_Memory: entity work.Instruction_Memory_VHDL
      port map      (      pc=> pc_current ,      instruction => instr  );
control: entity work.control_unit_VHDL
   port map   (reset => reset,    opcode => instr(15 downto 13),
    reg_dst => reg_dst,    mem_to_reg => mem_to_reg,
    alu_op => alu_op,    jump => jump,    branch => branch,    mem_read => mem_read,
    mem_write => mem_write,    alu_src => alu_src,
    reg_write => reg_write,    sign_or_zero => sign_or_zero  );
```

Note however PC(k downto 1) would be used to address 2k 2-byte-wides-words of InstrROM**



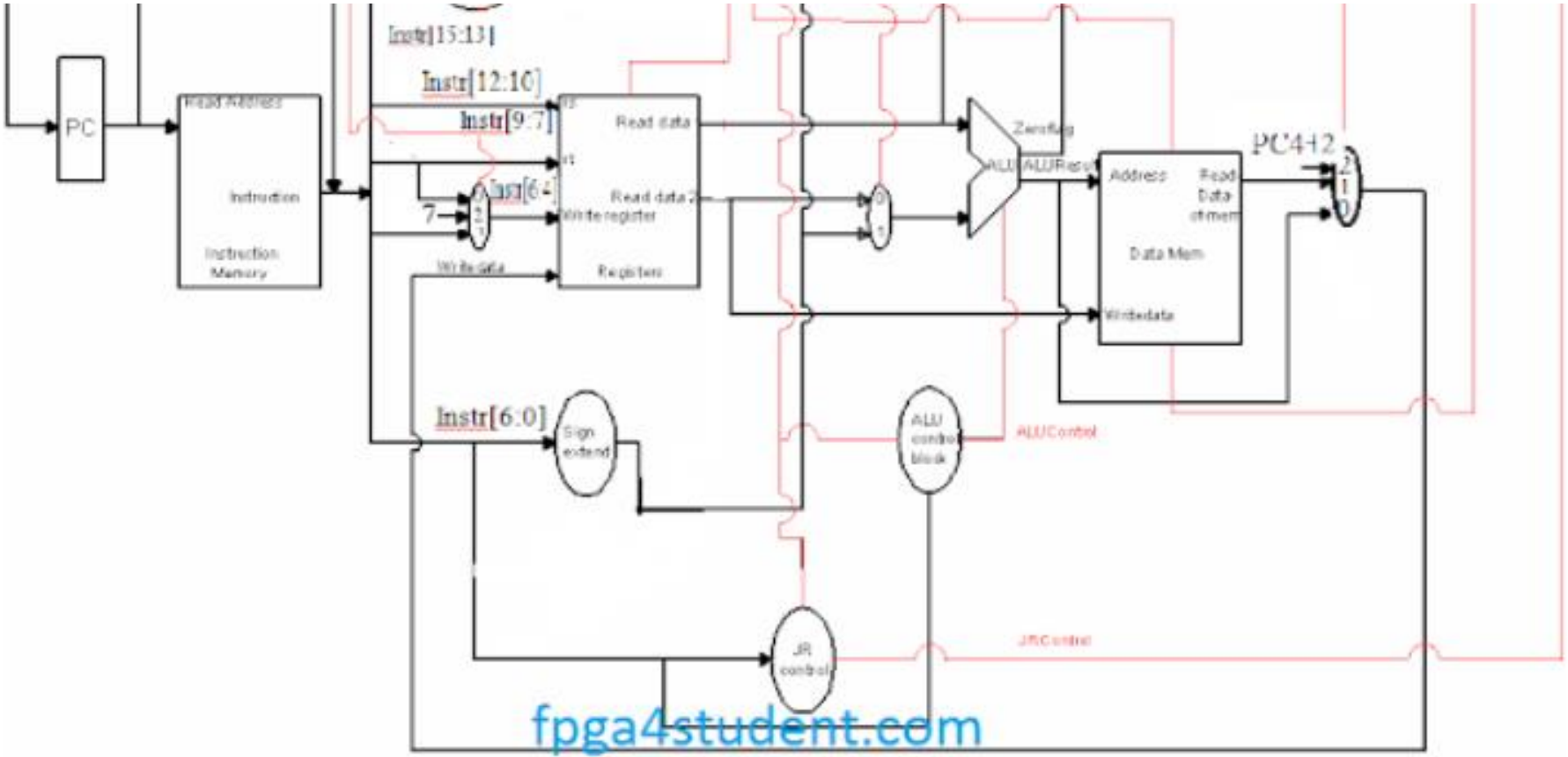

```
Instruction_Memory: entity work.Instruction_Memory_VHDL
    port map      (      pc=> pc_current ,      instruction => instr );
control: entity work.control_unit_VHDL
  port map   (reset => reset,    opcode => instr(15 downto 13),
   reg_dst => reg_dst,    mem_to_reg => mem_to_reg,
   alu_op => alu_op,    jump => jump,    branch => branch,    mem_read => mem_read,
   mem_write => mem_write,    alu_src => alu_src,
   reg_write => reg_write,    sign_or_zero => sign_or_zero  );
```

```
entity register_file_VHDL is port (  clk,rst: in std_logic; reg_write_en: in std_logic;
 reg_write_dest: in std_logic_vector(2 downto 0);
 reg_write_data: in std_logic_vector(15 downto 0); reg_read_addr_1: in std_logic_vector(2 downto 0);
 reg_read_data_1: out std_logic_vector(15 downto 0); reg_read_addr_2: in std_logic_vector(2 downto 0);
 reg_read_data_2: out std_logic_vector(15 downto 0)
);
end register_file_VHDL;
architecture Behavioral of register_file_VHDL is
    type reg_type is array (0 to 7 ) of std_logic_vector (15 downto 0);
    signal reg_array: reg_type;
begin
   process(clk,rst)  begin       if(rst='1') then  ……................
      elsif(rising_edge(clk)) then
         if(reg_write_en='1') then
            reg_array(to_integer(unsigned(reg_write_dest))) <= reg_write_data;
         end if;
      end if;
   end process;
   reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else
                                                        reg_array(to_integer(unsigned(reg_read_addr_1)));
   reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else
                                                        reg_array(to_integer(unsigned(reg_read_addr_2)));
end Behavioral;
```

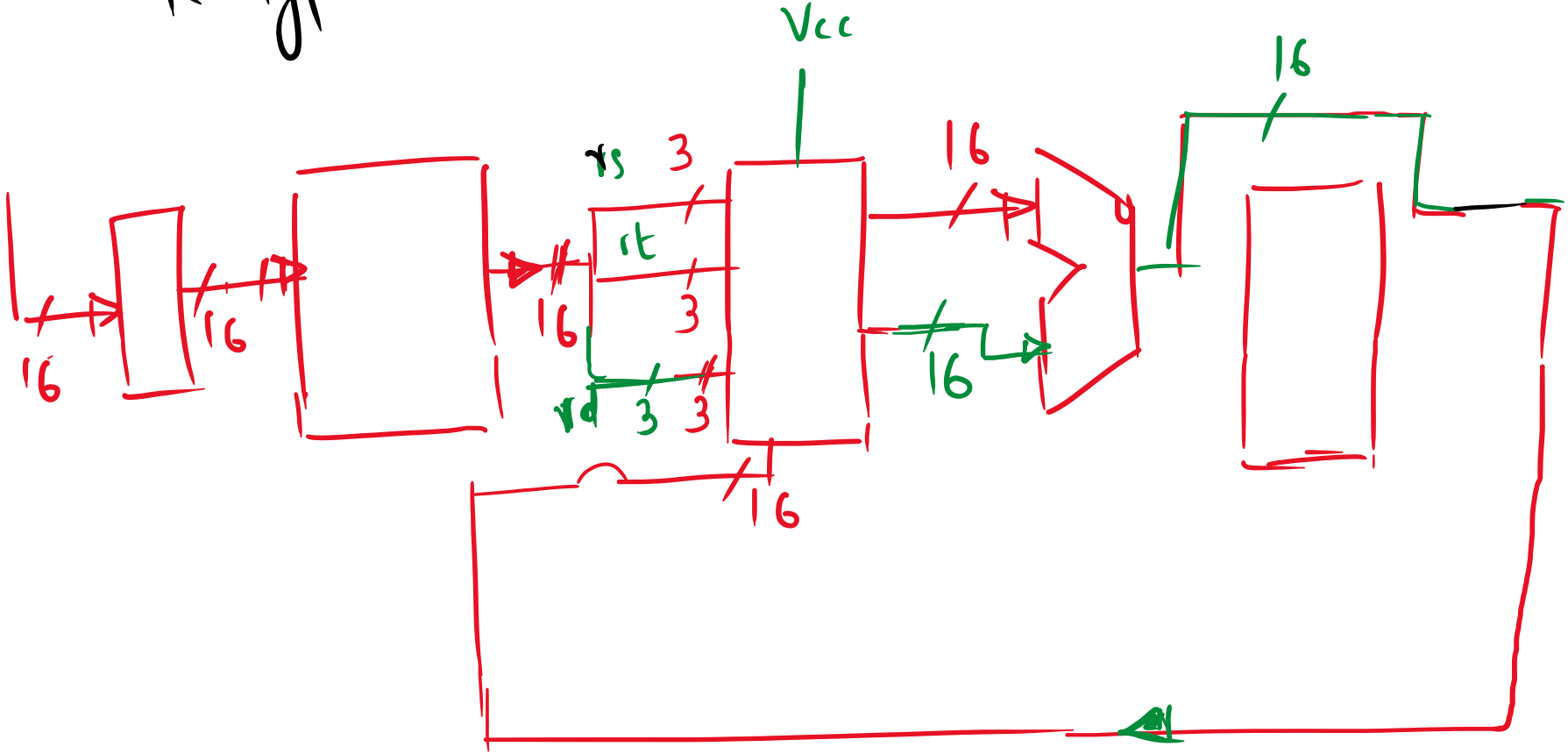


Walking along the Datapath

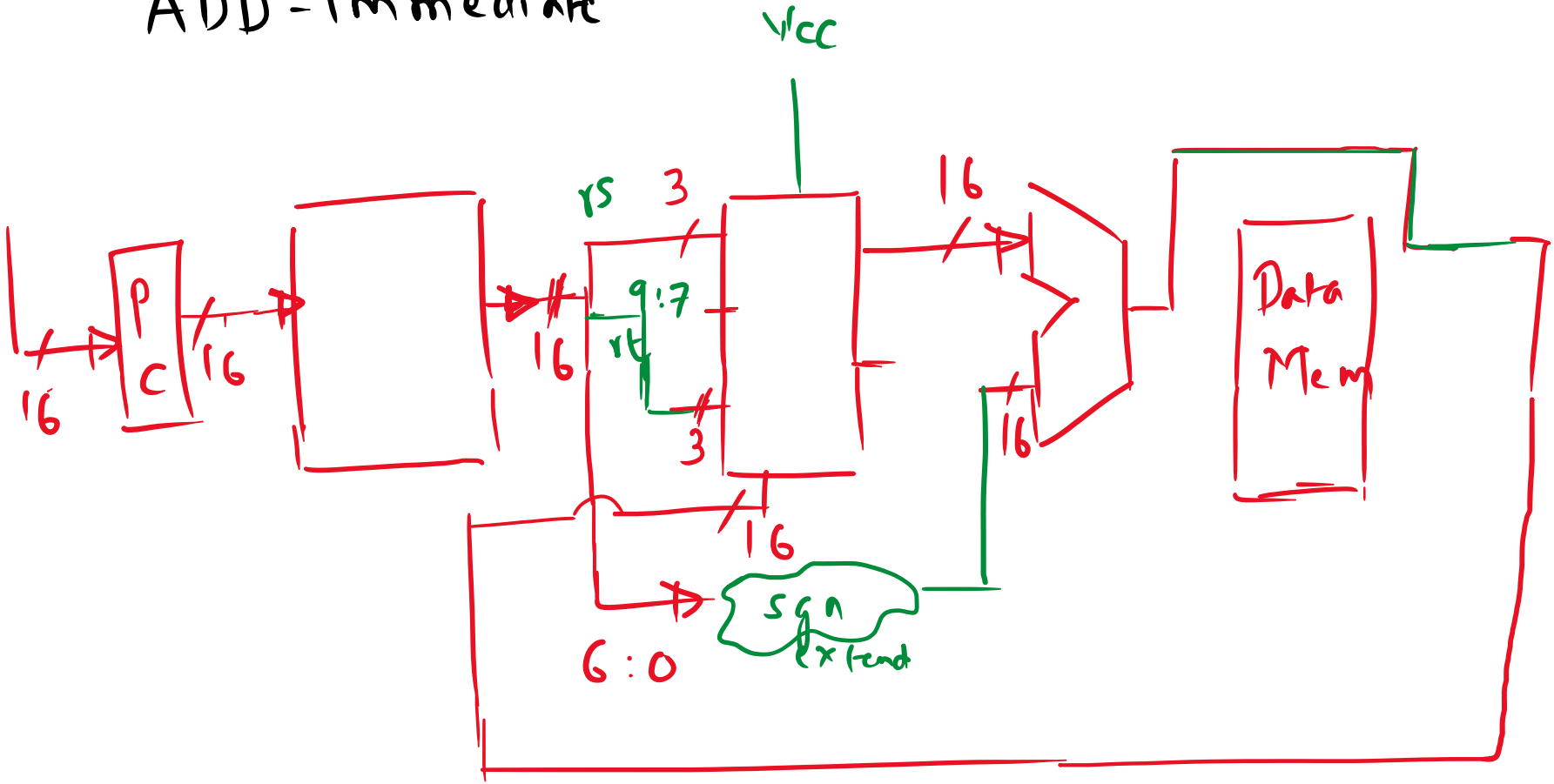
To help focus on the arithm/memory instructions, I have omitted branch/jump related microarchitecture from the diagram



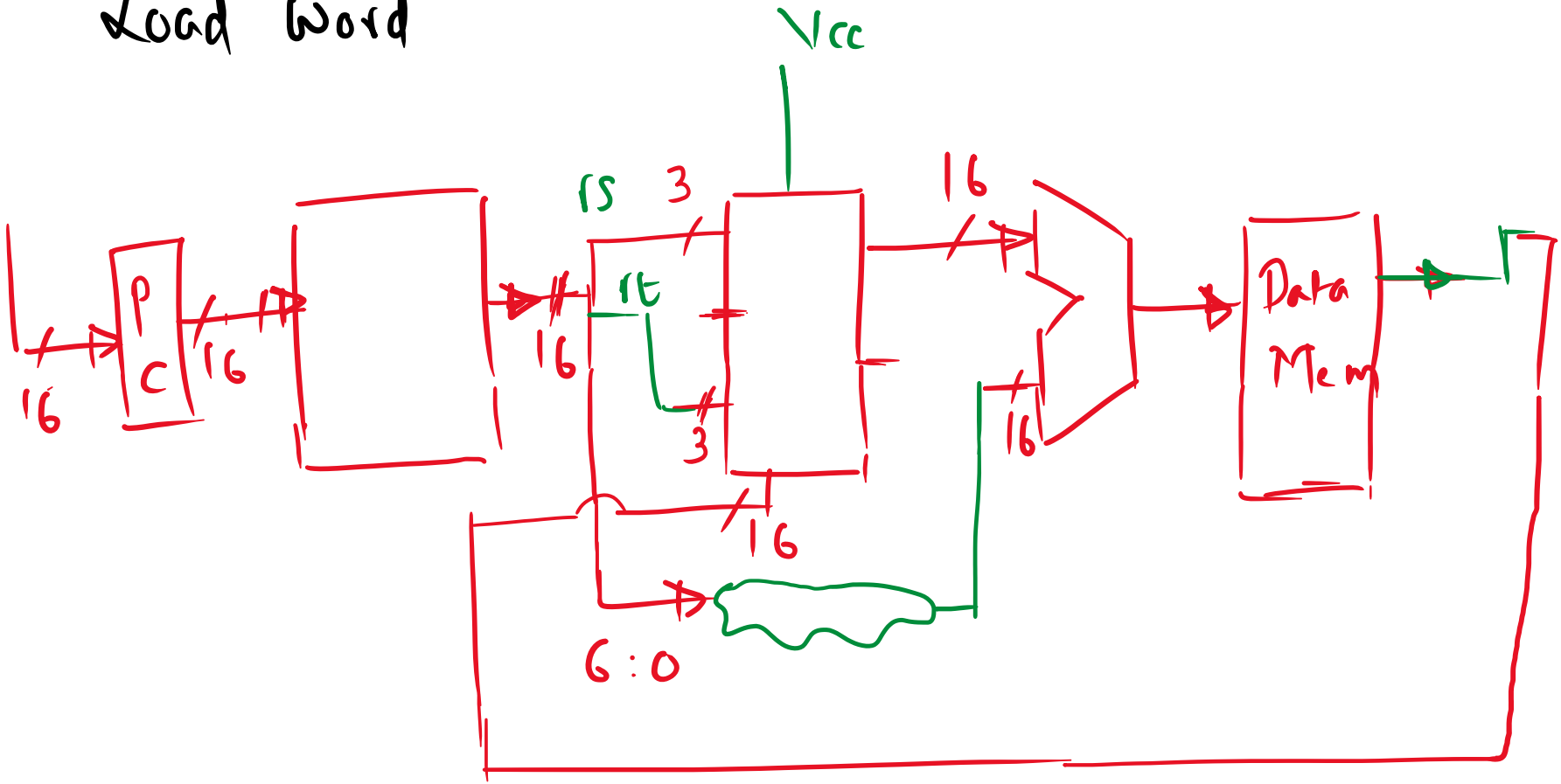
R-type Instruction



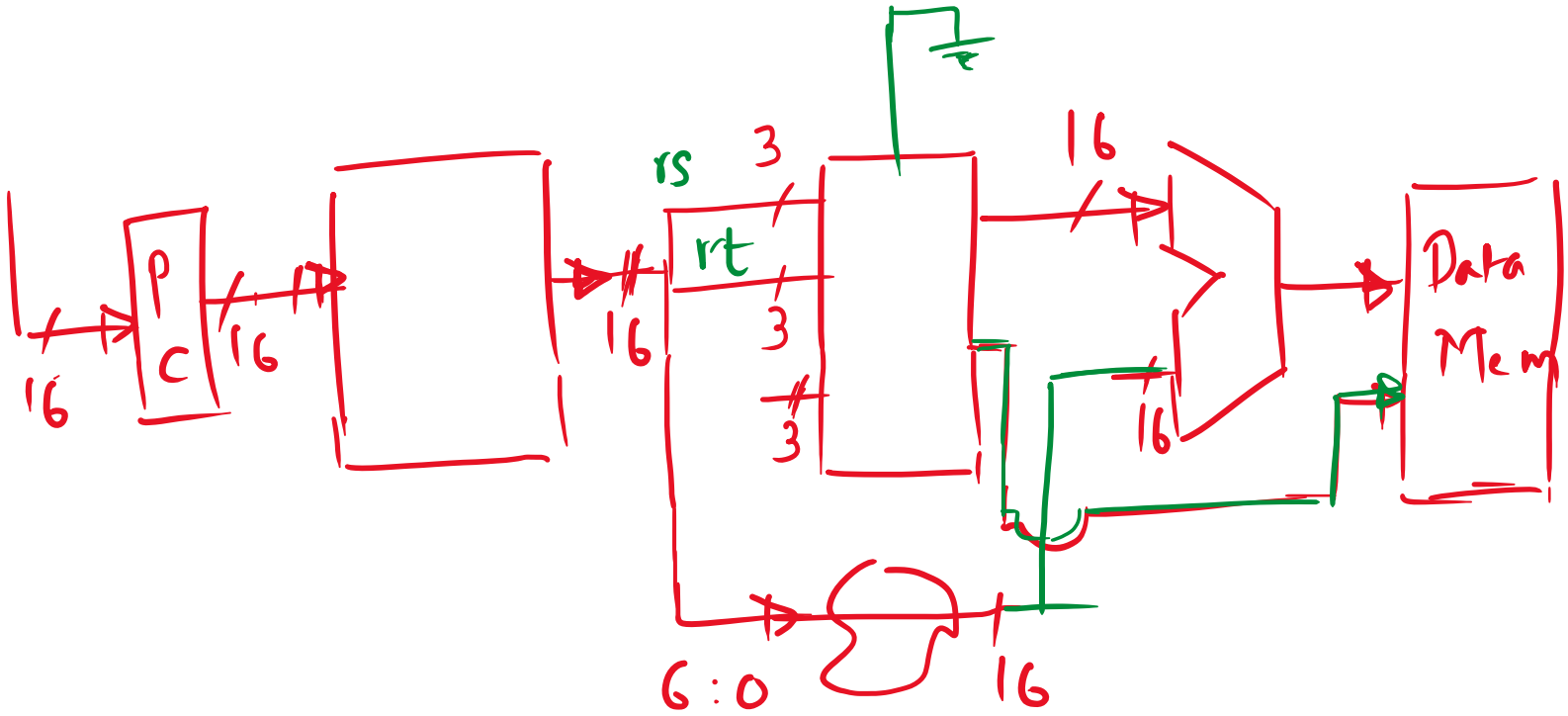
ADD - immediate



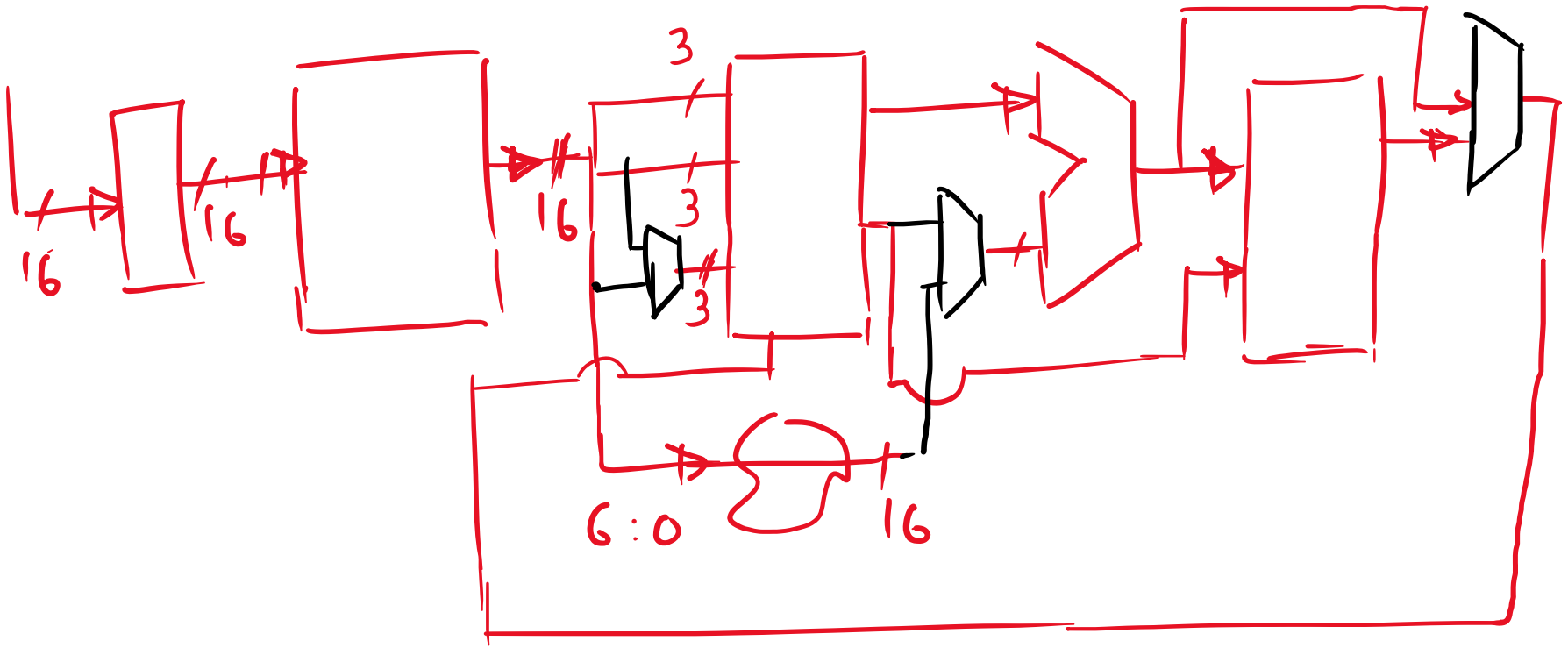
Load Word

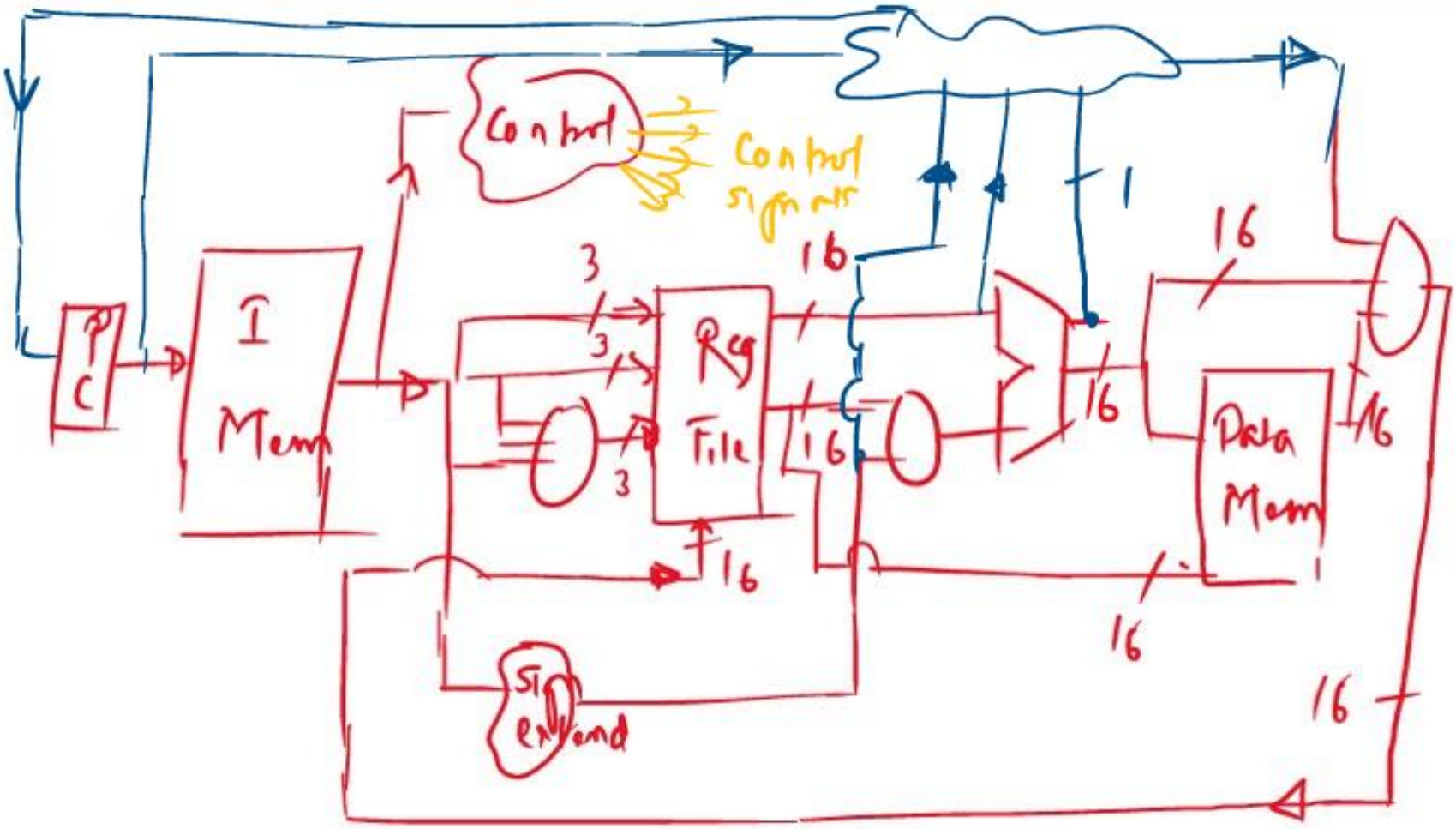


Store Word



Juxtaposing ...




Control
Control signals
1
16
3
3
3
16
PC
I Mem
Reg File
16
16
16
Data Mem
16
16
16
Sign extend

1. **Add : R[rd] = R[rs] + R[rt]**
2. **Subtract : R[rd] = R[rs] - R[rt]**
3. **And: R[rd] = R[rs] & R[rt]**
4. **Or : R[rd] = R[rs] | R[rt]**
5. **SLT: R[rd] = 1 if R[rs] < R[rt] else 0**
6. **Jr: PC=R[rs]**
7. **Lw: R[rt] = M[R[rs]+SignExtImm]**
8. **Sw : M[R[rs]+SignExtImm] = R[rt]**
9. **Beq : if(R[rs]==R[rt]) PC=PC+1+BranchAddr**
10. **Addi: R[rt] = R[rs] + SignExtImm**
11. **J : PC=JumpAddr**
12. **Jal : R[7]=PC+2;PC=JumpAddr**
13. **SLTI: R[rt] = 1 if R[rs] < imm else 0**

SignExtImm = { 9{immediate[6]}, imm

JumpAddr = { (PC+1)[15:13], address}

BranchAddr = { 7{immediate[6]}, immediate, 1'b0 }

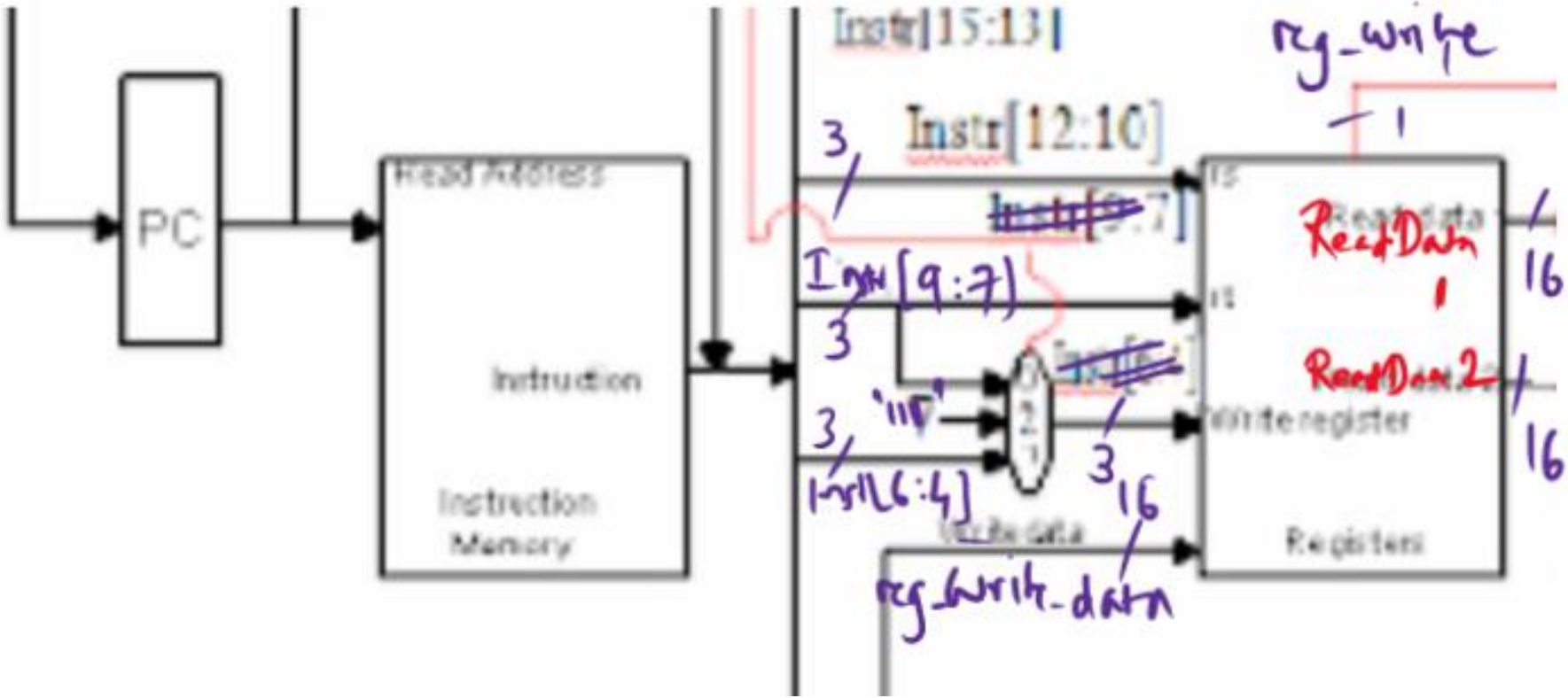
RTN

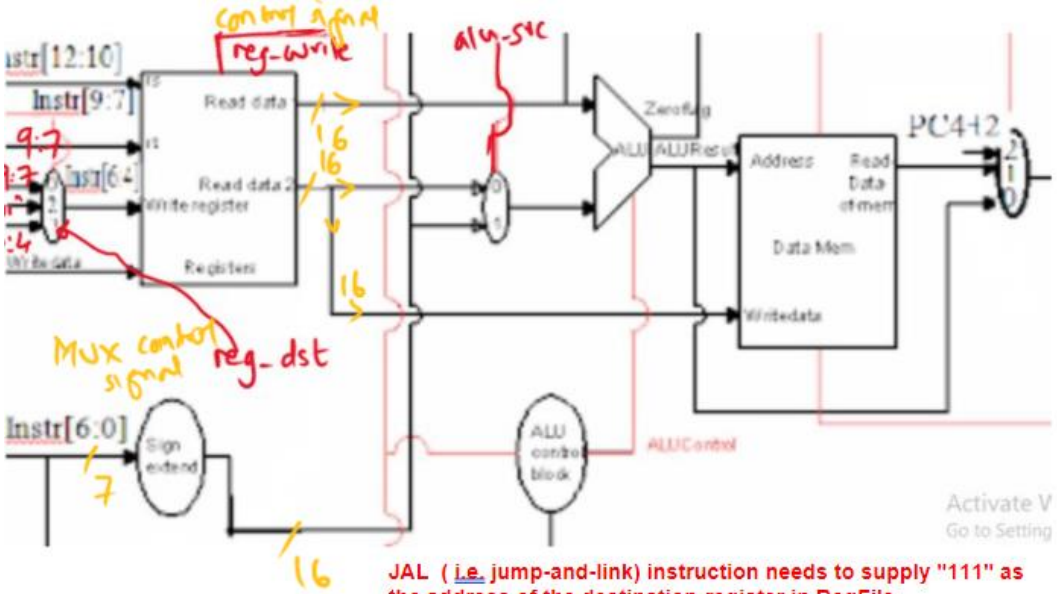
Register Transfer Notation

for description of Datapath

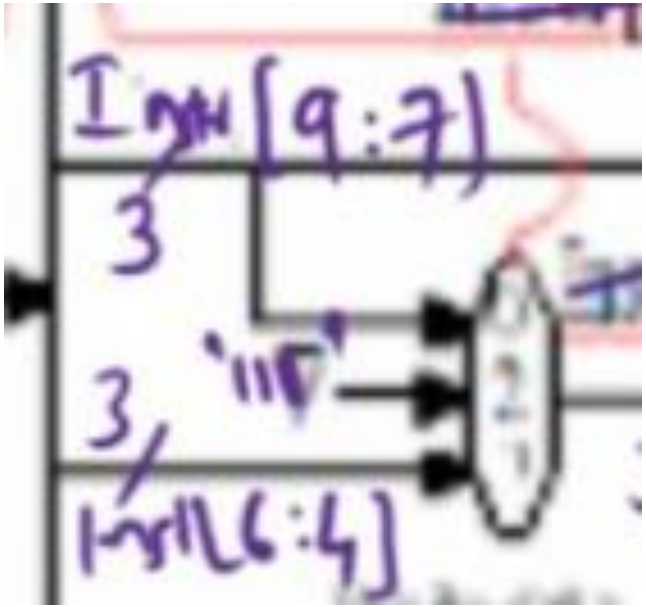
Caution : Mismatch between this RTN description and the implementation. Here PC+1 Indicates the address of next instruction-word (not byte). Whereas in HDL code, it is assumed that PC contains address of first byte of instruction

Note the corrections : The rs field of Instr, i.e. Instr[12:10] is fed to reg_read_addr_1 port of RegFile, rt field, i.e. Instr[9:7] drives the 2nd address port of RegFile. And both rt and rd (i.e. Instr[6:4]) fields of Instr, along with constant "111" are options for the 3rd address port of RegFile



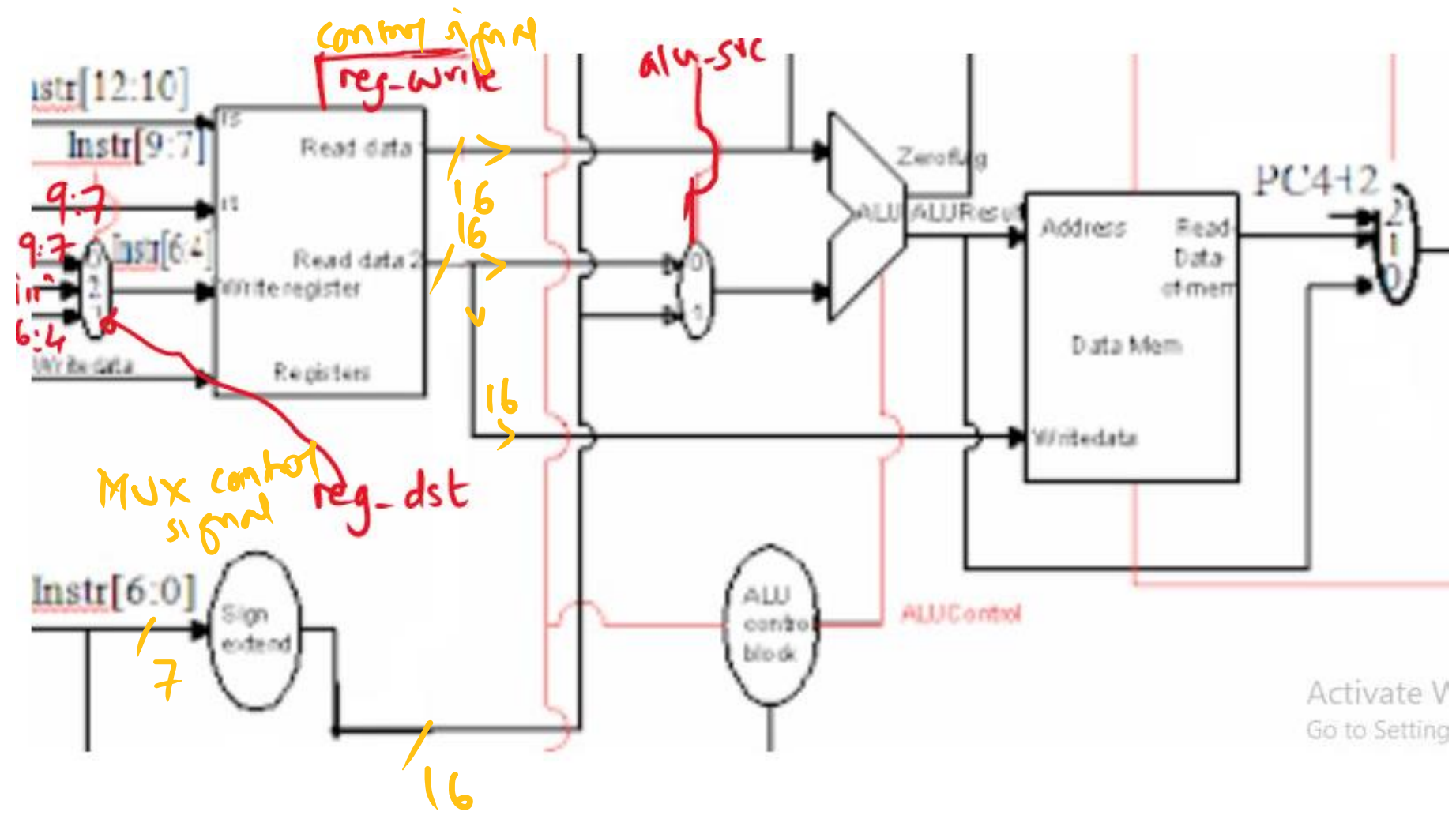


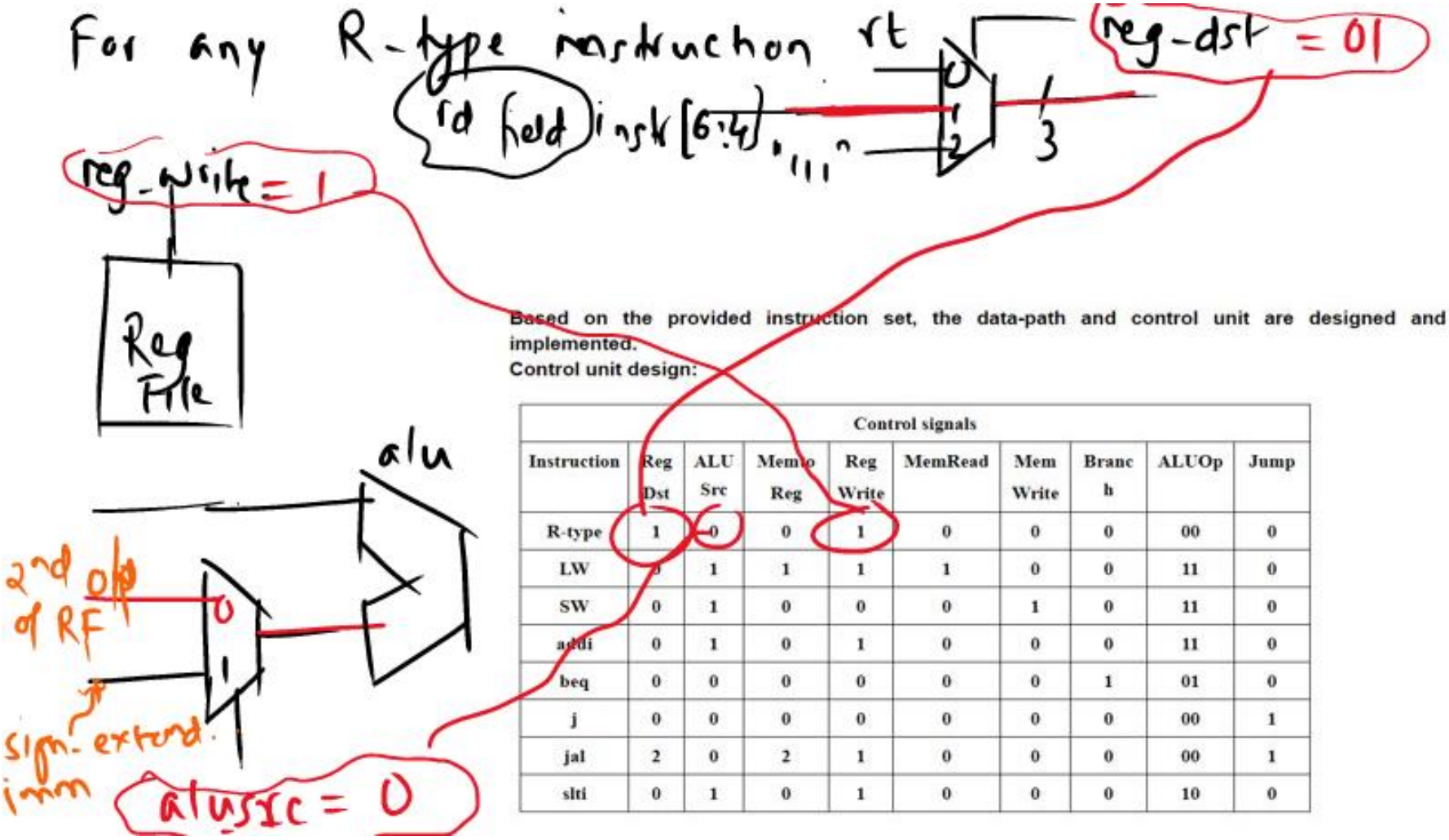
JAL (i.e. jump-and-link) instruction needs to supply "111" as the address of the destination-register in RegFile



JAL (i.e. jump-and-link) insruction needs to supply "111" as the address of the destination-register in RegFile

Let's Look at Some Control Signals !


control signal
reg-write
alu-src
Instr[12:10]
Instr[9:7]
9:7
Instr[6:4]
6:4
Read data
Read data 2
Write register
Registers
16
16
16
Zero flag
ALU
ALU Result
Address
Read Data
Data Mem
Write data
PC+2
MUX control signal
reg_dst
Instr[6:0]
Sign extend
7
16
ALU control block
ALU Control
Activate W
Go to Setting



Based on the provided instruction set, the data-path and control unit are designed and implemented.

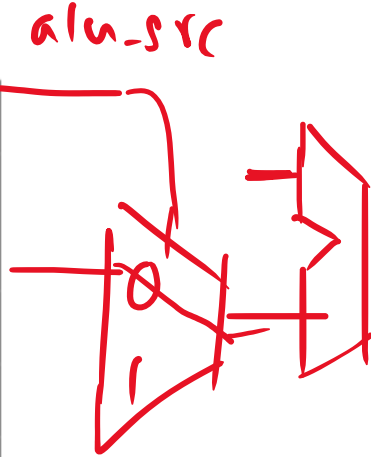
Control unit design:

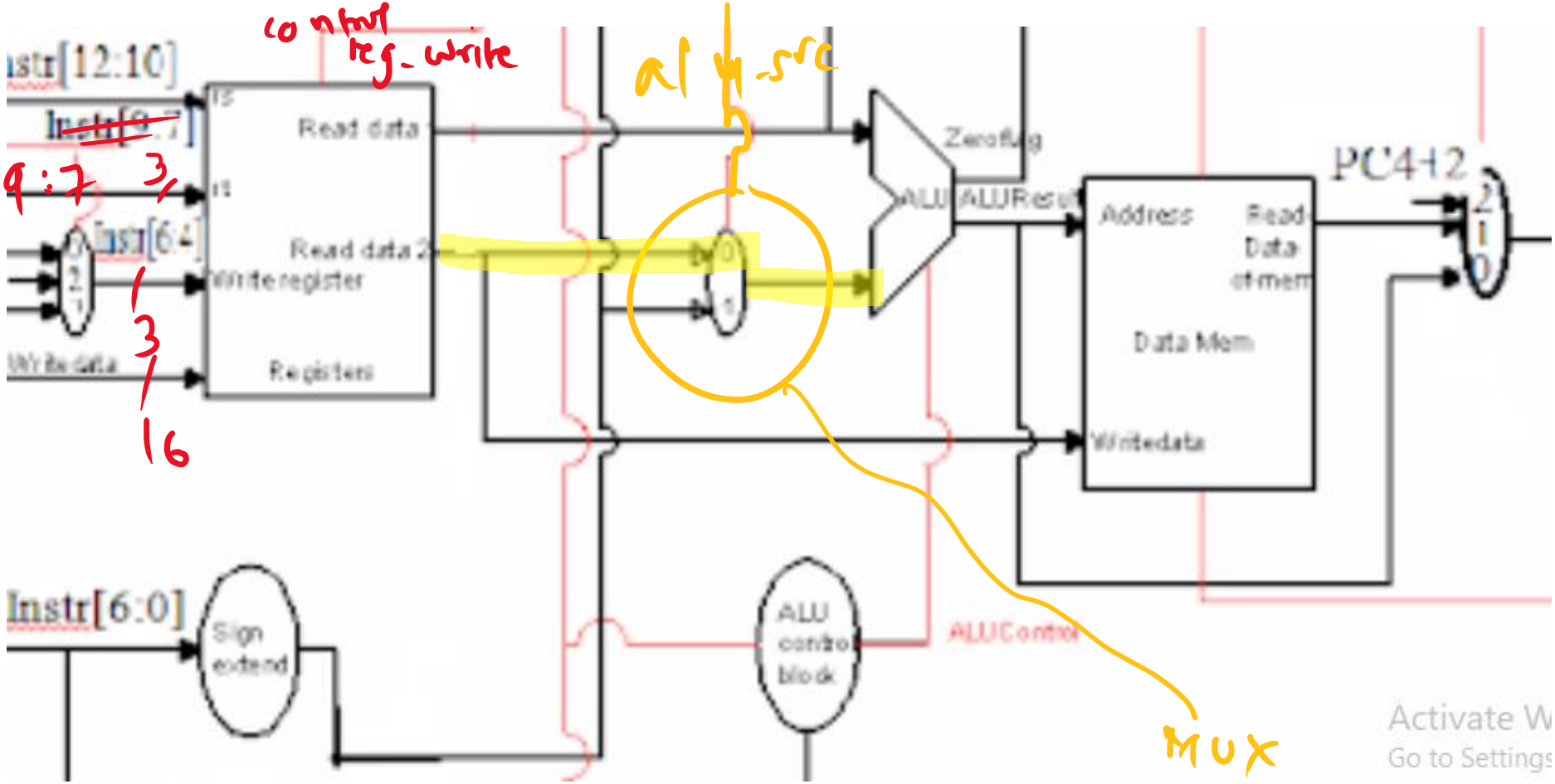
	Control signals								
Instruction	Reg Dst	ALU Src	Mem to Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
R-type	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	11	0
SW	0	1	0	0	0	1	0	11	0
addi	0	1	0	1	0	0	0	11	0
beq	0	0	0	0	0	0	1	01	0
j	0	0	0	0	0	0	0	00	1
jal	2	0	2	1	0	0	0	00	1
slti	0	1	0	1	0	0	0	10	0

Based on the provided instruction set, the data-path and control unit are designed and implemented.

Control unit design:

Control signals									
Instruction	Reg Dst	ALU Src	Memto Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
R-type	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	11	0
SW	0	1	0	0	0	1	0	11	0
addi	0	1	0	1	0	0	0	11	0
beq	0	0	0	0	0	0	1	01	0
j	0	0	0	0	0	0	0	00	1
jal	2	0	2	1	0	0	0	00	1
slti	0	1	0	1	0	0	0	10	0

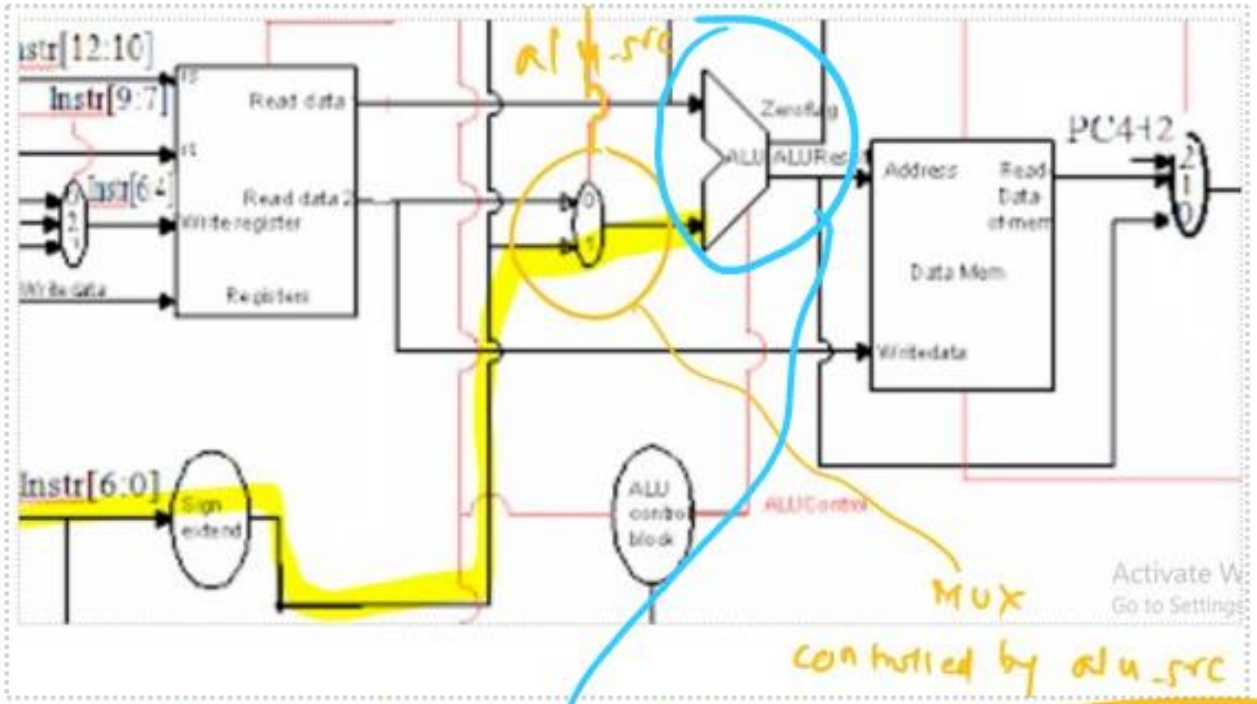



control
reg_write
alu_src
Instr[12:10]
Instr[9:7]
9:7
3
Instr[6:4]
3
16
Read data 1
Read data 2
Write register
Write data
Registers
Zero flag
ALU
ALU Result
Address
Read data
Data Mem
Write data
PC+4
2
1
0
Instr[6:0]
Sign extend
ALU control block
ALUControl
MUX
controlled by alu_src
Activate W
Go to Settings

Based on the provided instruction set, the data-path and control unit are designed and implemented.

Control unit design:

Control signals									
Instruction	Reg Dst	ALU Src	Memto Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
R-type	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	11	0
SW	0	1	0	0	0	1	0	11	0
addi	0	1	0	1	0	0	0	11	0
beq	0	0	0	0	0	0	1	01	0
j	0	0	0	0	0	0	0	00	1
jal	2	0	2	1	0	0	0	00	1
slti	0	1	0	1	0	0	0	10	0




```
read_data2 <= imm_ext when alu_src='1' else reg_read_data_2;
alu: entity work.ALU_VHDL port map (
  a => reg_read_data_1,   b => read_data2,
  alu_control => ALU_Control,   alu_result => ALU_out, zero => zero_flag );
```

MUX

ALU

```
--   We would need either sign-extended version of zero-extended version
--        of "imm" field ( i.e. instr(6 downto 0) ) for I-format arithmetic (add/sub)
                                                    or I-format logical-op instruction
tmp1 <= (others => instr(6));

sign_ext_im <= tmp1 & instr(6 downto 0);

zero_ext_im <= "000000000"& instr(6 downto 0);

imm_ext <= sign_ext_im when sign_or_zero='1' else zero_ext_im;


--  this sign/zero extended version of instr(6 downto 0) gets routed to the
--      2nd input of ALU as an alternative to the 2nd 16-bit-wide output of RF

read_data2 <= imm_ext when alu_src='1' else reg_read_data_2;
```

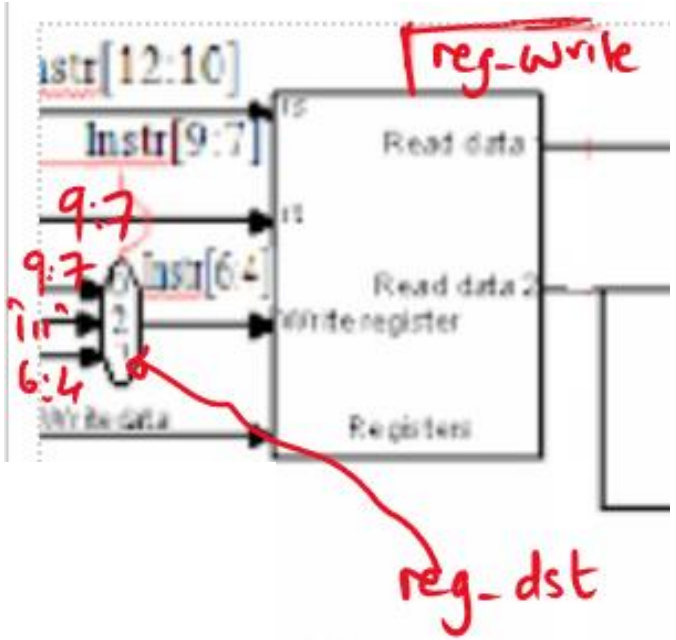
Based on the provided instruction set, the data-path and control unit are designed and implemented.

Control unit design:

Control signals									
Instruction	Reg Dst	ALU Src	Memto Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
R-type	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	11	0
SW	0	1	0	0	0	1	0	11	0
addi	0	1	0	1	0	0	0	11	0
beq	0	0	0	0	0	0	1	01	0
j	0	0	0	0	0	0	0	00	1
jal	2	0	2	1	0	0	0	00	1
slti	0	1	0	1	0	0	0	10	0

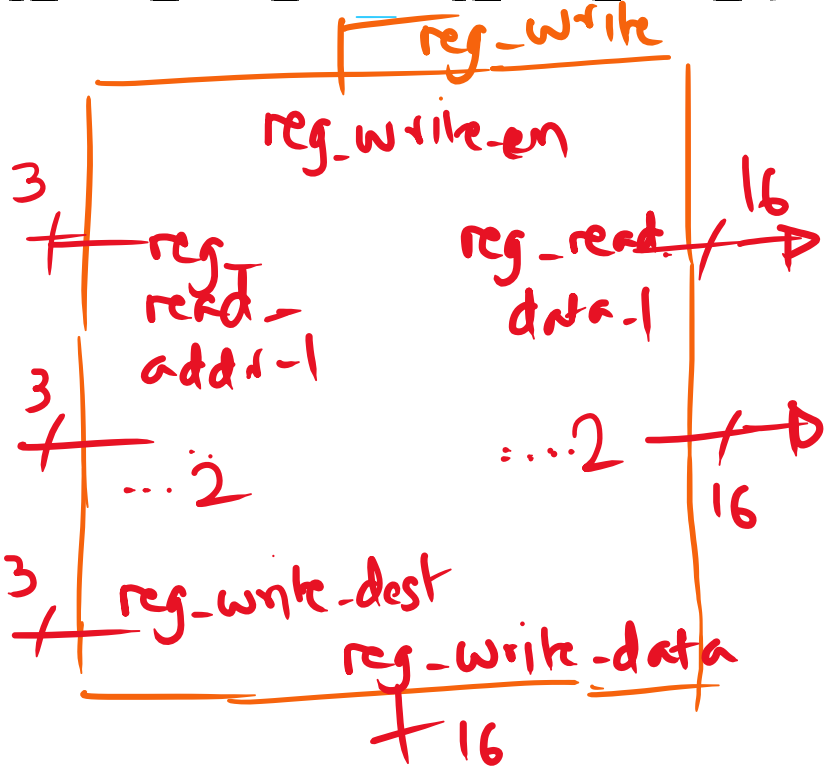
A Closer Look at the Register File in this VHDL model

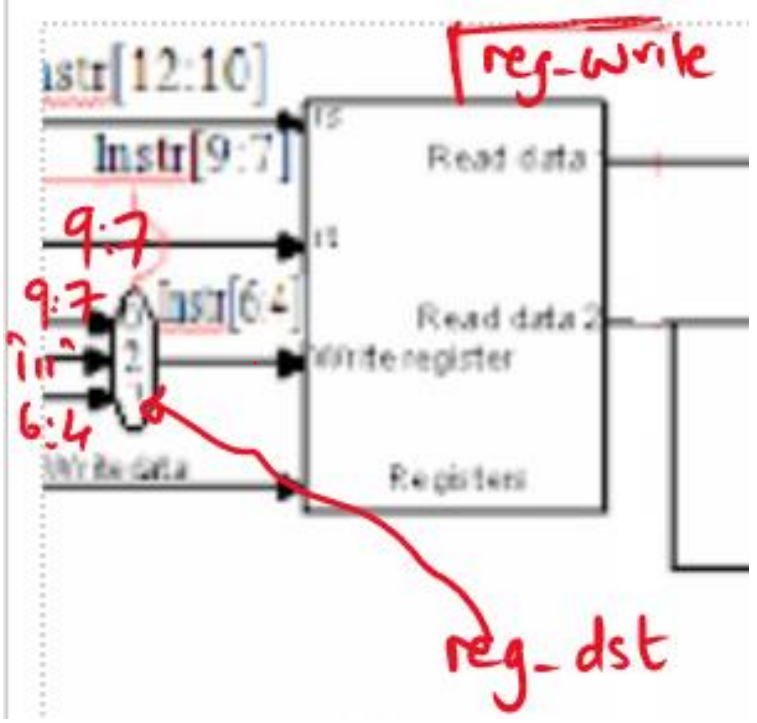
```
entity register_file_VHDL is port (  clk,rst: in std_logic;  reg_write_en: in std_logic;
 reg_write_dest: in std_logic_vector(2 downto 0);
 reg_write_data: in std_logic_vector(15 downto 0);  reg_read_addr_1: in std_logic_vector(2 downto 0);
 reg_read_data_1: out std_logic_vector(15 downto 0);  reg_read_addr_2: in std_logic_vector(2 downto 0);
 reg_read_data_2: out std_logic_vector(15 downto 0)
);
```



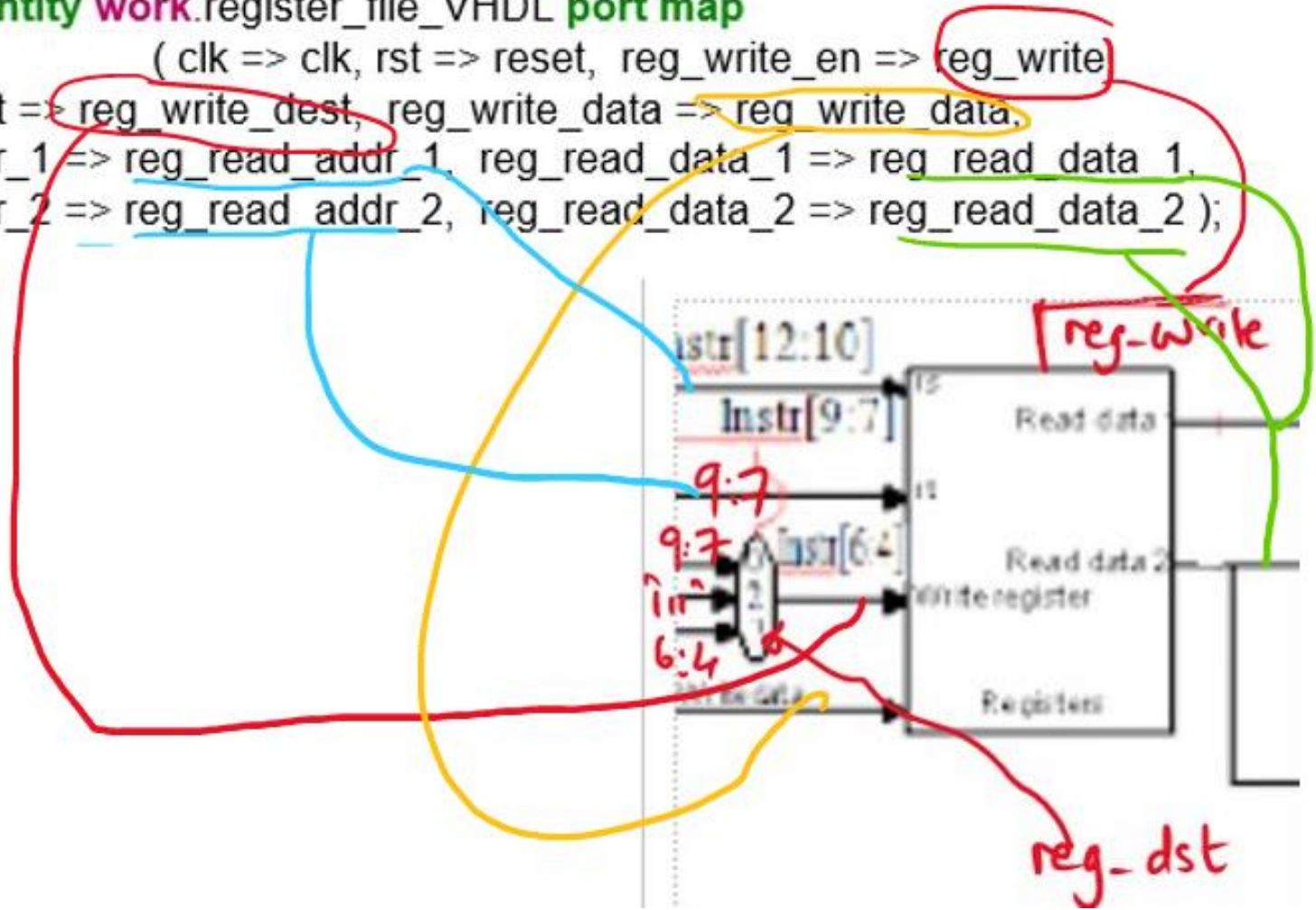

```
architecture Behavioral of register_file_VHDL is
   type reg_type is array (0 to 7 ) of std_logic_vector (15 downto 0);
   signal reg_array: reg_type;
begin
  process(clk,rst)  begin      if(rst='1') then  ..........................
     elsif(rising_edge(clk)) then
        if(reg_write_en='1') then
           reg_array(to_integer(unsigned(reg_write_dest))) <= reg_write_data;
        end if;
     end if;
  end process;
  reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else
                                                    reg_array(to_integer(unsigned(reg_read_addr_1)));
  reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else
                                                    reg_array(to_integer(unsigned(reg_read_addr_2)));
end Behavioral;
```

```
register_file: entity work.register_file_VHDL port map
                                ( clk => clk, rst => reset,  reg_write_en => reg_write,
reg_write_dest => reg_write_dest,  reg_write_data => reg_write_data,
reg_read_addr_1 => reg_read_addr_1,  reg_read_data_1 => reg_read_data_1,
reg_read_addr_2 => reg_read_addr_2,  reg_read_data_2 => reg_read_data_2 );
```

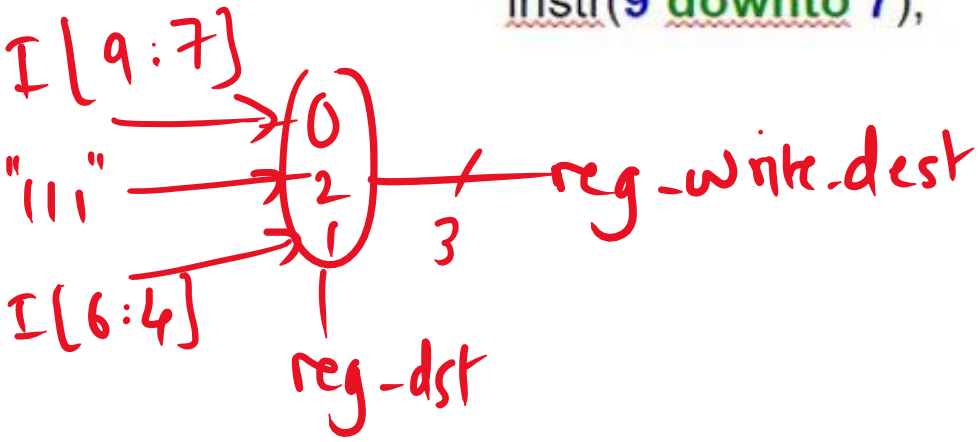


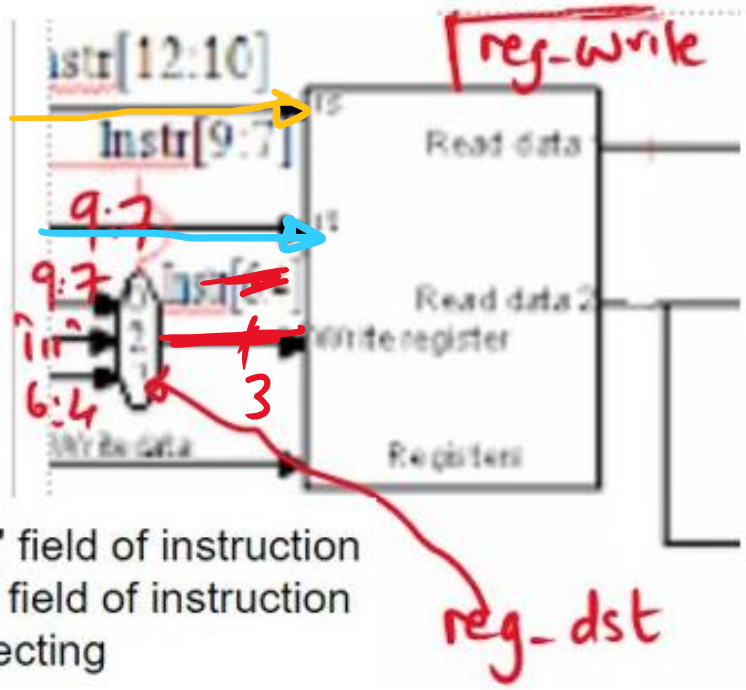


```
register_file: entity work.register_file_VHDL port map
                    ( clk => clk, rst => reset,  reg_write_en => reg_write,
reg_write_dest => reg_write_dest,  reg_write_data => reg_write_data,
reg_read_addr_1 => reg_read_addr_1,  reg_read_data_1 => reg_read_data_1,
reg_read_addr_2 => reg_read_addr_2,  reg_read_data_2 => reg_read_data_2 );
```



```
reg_write_dest <= "111" when reg_dst= "10" else
                  instr(6 downto 4) when reg_dst= "01" else
                  instr(9 downto 7);
```

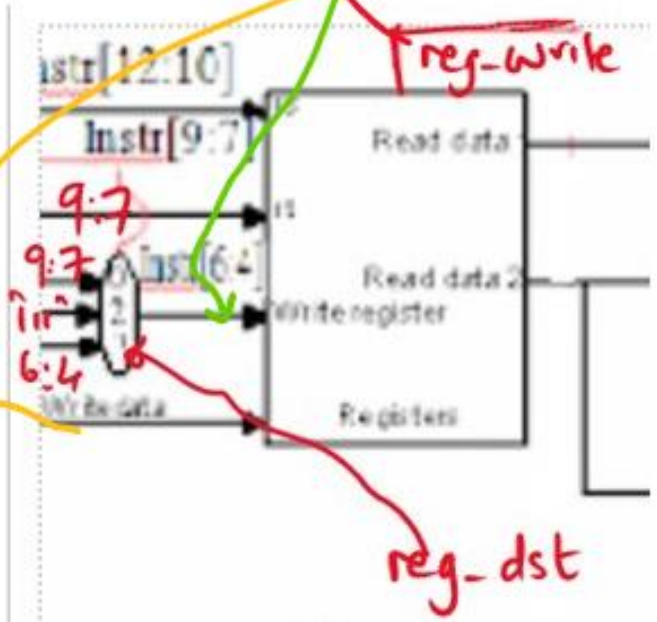




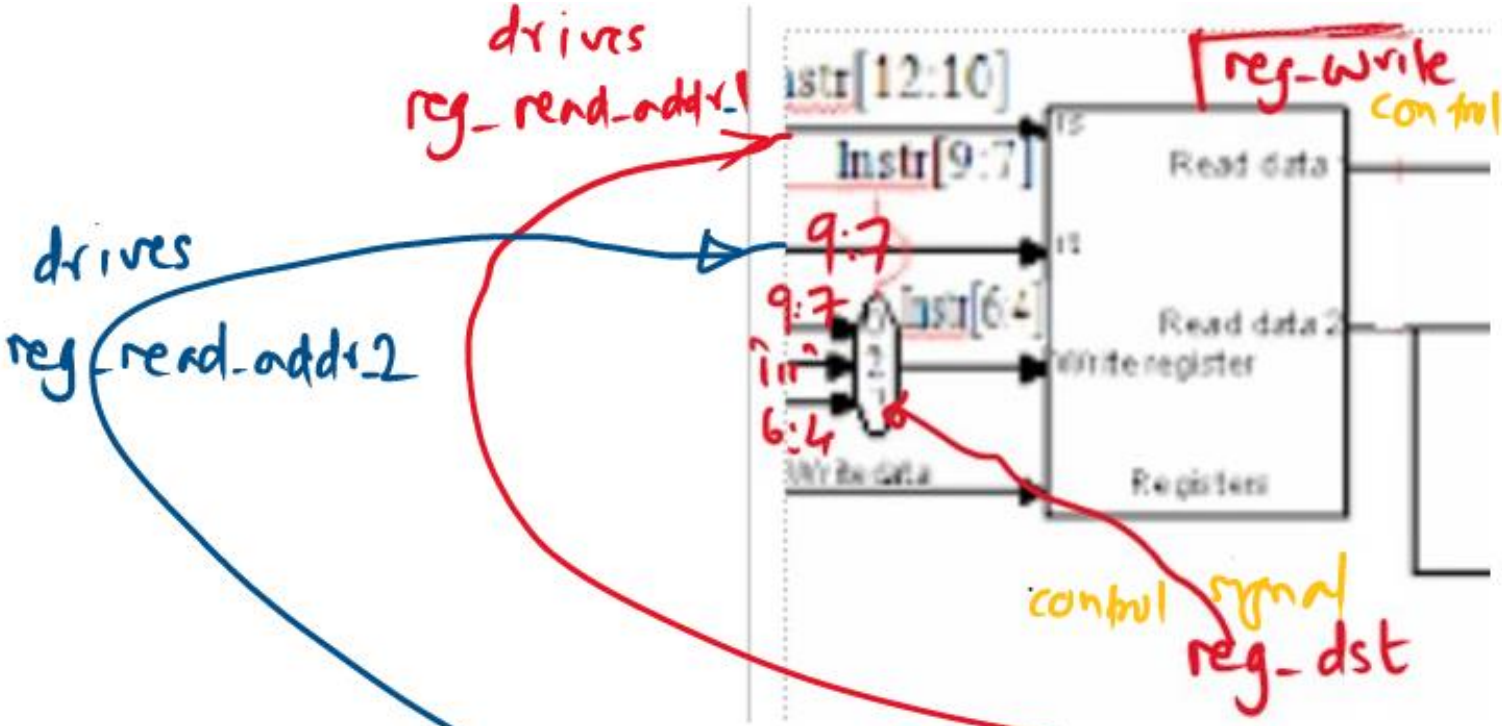

```
reg_read_addr_1 <= instr(12 downto 10);     -- "rs" field of instruction
reg_read_addr_2 <= instr(9 downto 7);       -- "rt" field of instruction
   --  this pair of 3-bit wide signals are used for selecting
   --      2 registers from RegFile  for read-out
   --          at the pair of 16-bit-wide output ports of the RegFile
```

```
reg_write_dest <= "111" when reg_dst= "10" else
                  instr(6 downto 4) when reg_dst= "01" else
                  instr(9 downto 7);

    --   as indicated during LecMeet-11  Thu-18-Feb21
    --   reg_write_dest is output of a 3-bit-wide 3-way multiplexer
         -- it chooses either instr(9 downto 7)  ( namely, the "rt" field of instruction )
         --                or      instr(6 downto 4)  ( namely, the "rd" field )
         --                or      "111"  respectively for
         --     the following settings of "reg_dst" , namely "00","01","10"


reg_read_addr_1 <= instr(12 downto 10);     -- "rs" field of instruction
reg_read_addr_2 <= instr(9 downto 7);       -- "rt" field of instruction
   --  this pair of 3-bit wide signals are used for selecting
   --      2 registers from RegFile  for read-out
   --          at the pair of 16-bit-wide output ports of the RegFile
```

```
process(clk,rst) begin      if(rst='1') then .....................
   elsif(rising_edge(clk)) then
      if(reg_write_en='1') then
         reg_array(to_integer(unsigned(reg_write_dest))) <= reg_write_data;
      end if;
   end if;
end process;
```



vhdl modeling of write operation inside RegFile

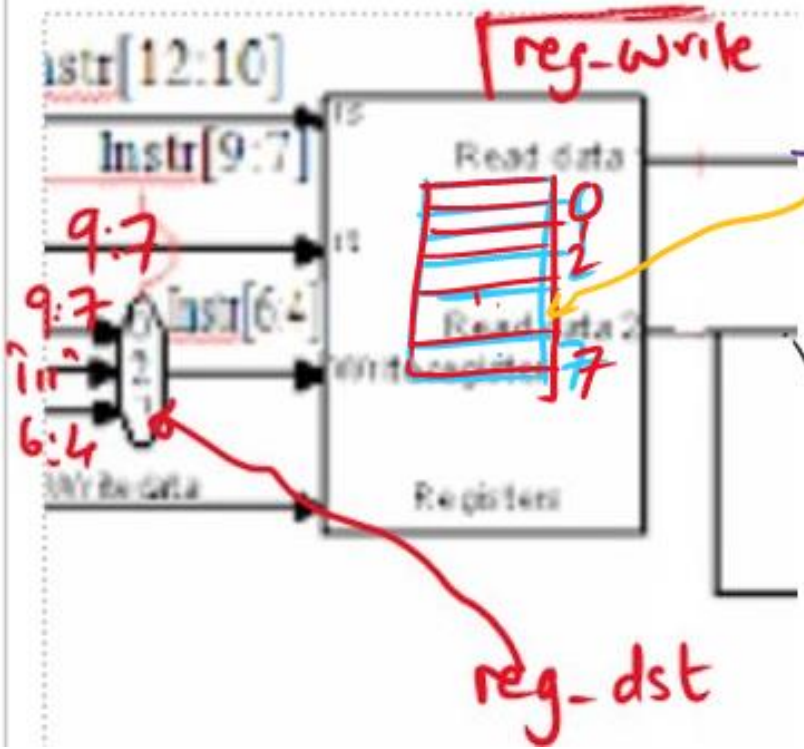



```
reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else
                                    reg_array(to_integer(unsigned(reg_read_addr_1)));
reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else
                                    reg_array(to_integer(unsigned(reg_read_addr_2)));
```

```
reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else
                   reg_array(to_integer(unsigned(reg_read_addr_1)));
reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else
                   reg_array(to_integer(unsigned(reg_read_addr_2)));
```

array of registers

The Read Out logic of Register File

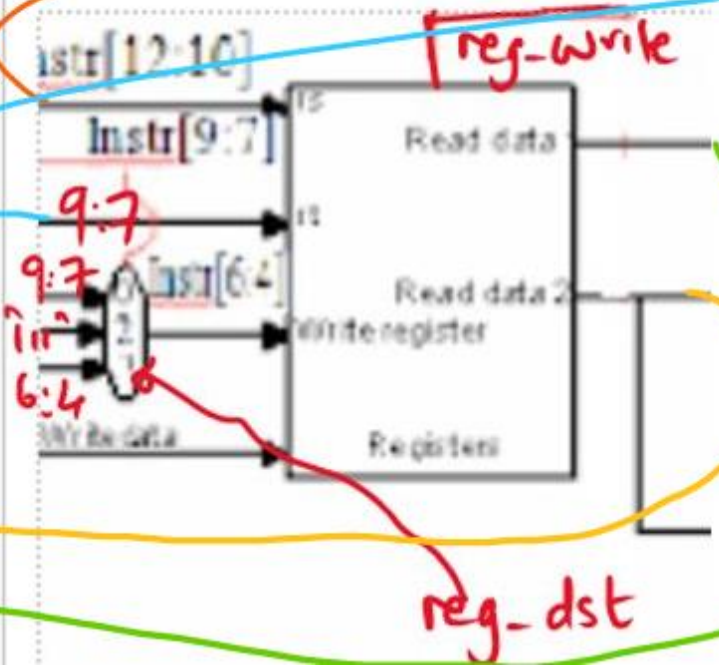


driven by reg_read_data_1

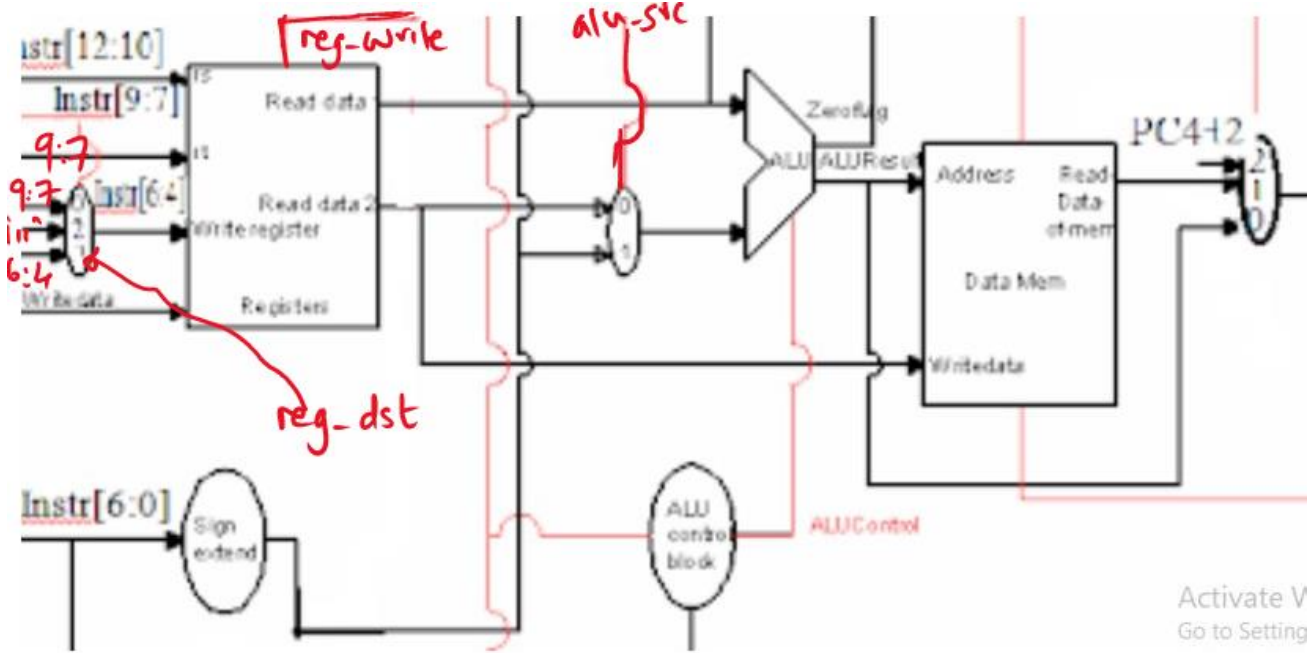
driven by reg_read_data_2

```
reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else
                   reg_array(to_integer(unsigned(reg_read_addr_1)));
reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else
                   reg_array(to_integer(unsigned(reg_read_addr_2)));
```

array of registers

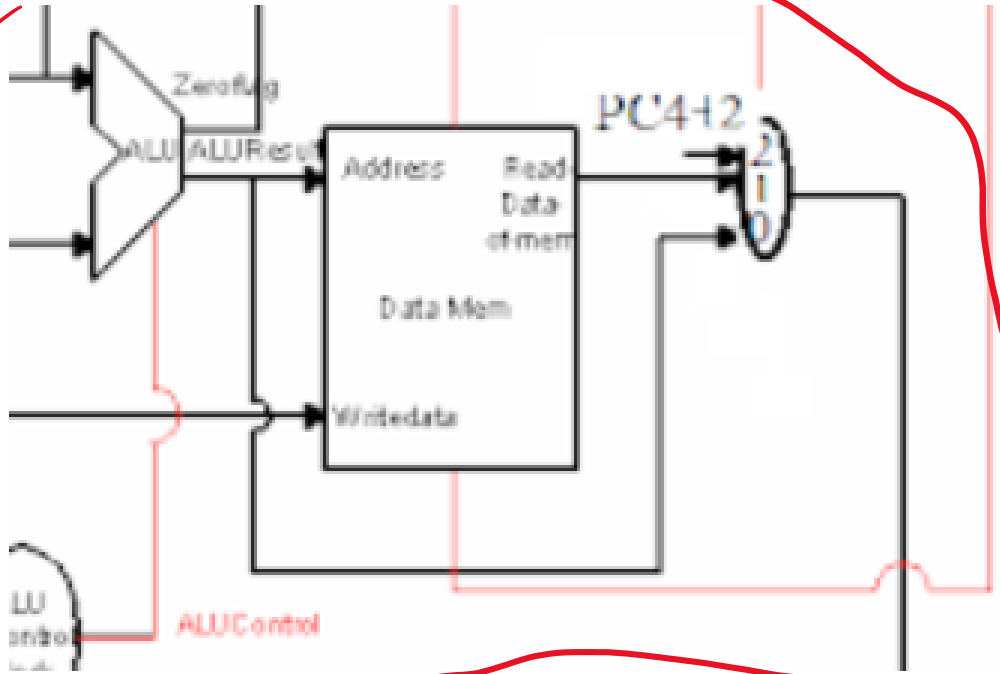


WriteBack Portion of Datapath

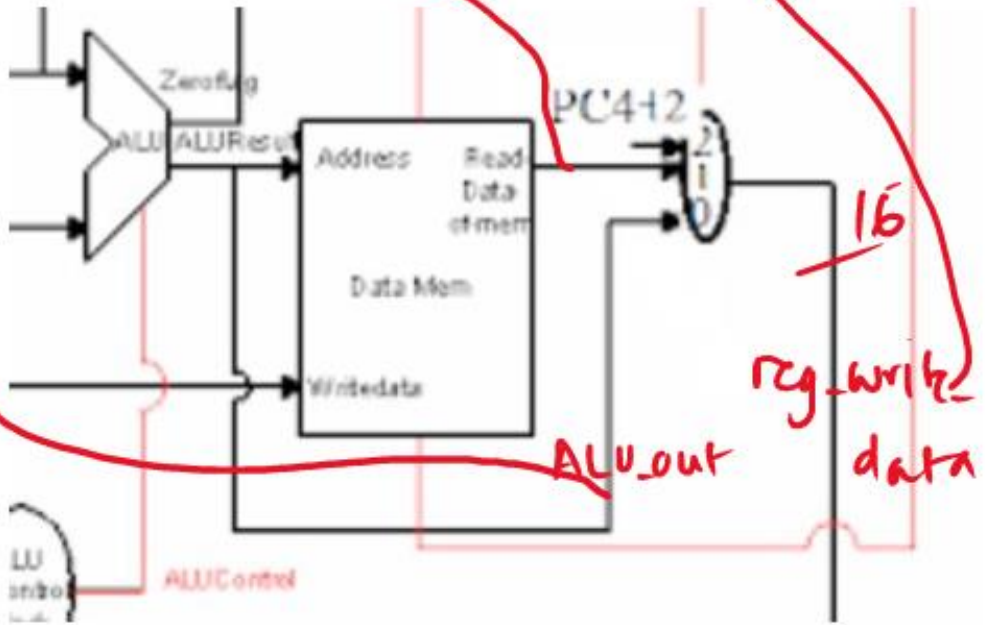

reg_write
alu_src
Instr[12:10]
Instr[9:7]
9:7
Instr[6:4]
6:4
Read data
Read data 2
Write register
Registers
reg_dst
Zero flag
ALU
ALU Result
Address
Read Data
Data Mem
Write data
PC+4
2
1
0
Instr[6:0]
Sign extend
ALU control block
ALU Control
Activate W
Go to Setting

```
-- write back of the MIPS Processor in VHDL
reg_write_data <= pc2 when (mem_to_reg = "10")  else
                  mem_read_data  when (mem_to_reg = "01") else
                  ALU_out;

pc_out <= pc_current;

alu_result <= ALU_out;
```

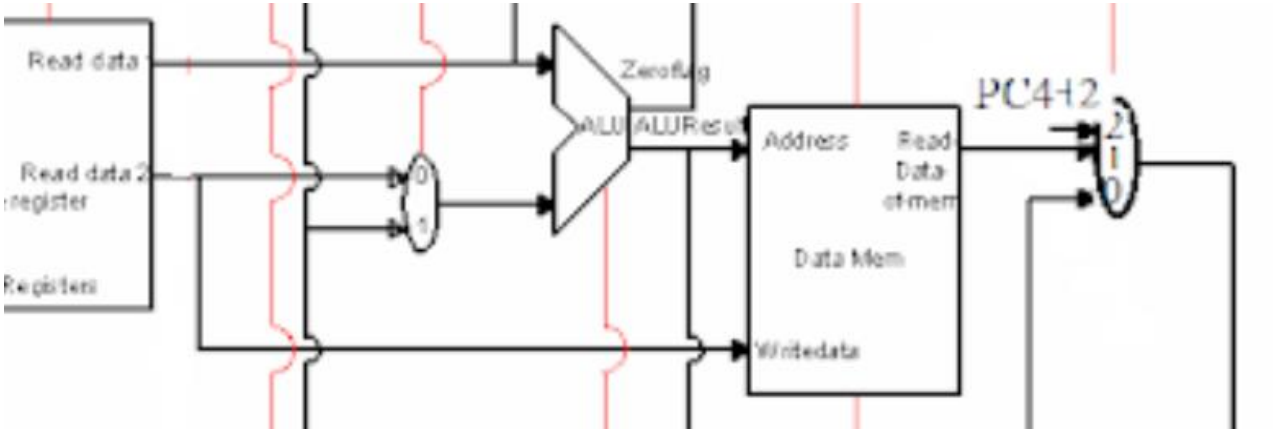
ignore



```
-- write back of the MIPS Processor in VHDL
reg_write_data <= pc2 when (mem_to_reg = "10")  else
                  mem_read_data  when (mem_to_reg = "01") else
                  ALU_out;

pc_out <= pc_current;

alu_result <= ALU_out;
```



Data Memory model in VHDL



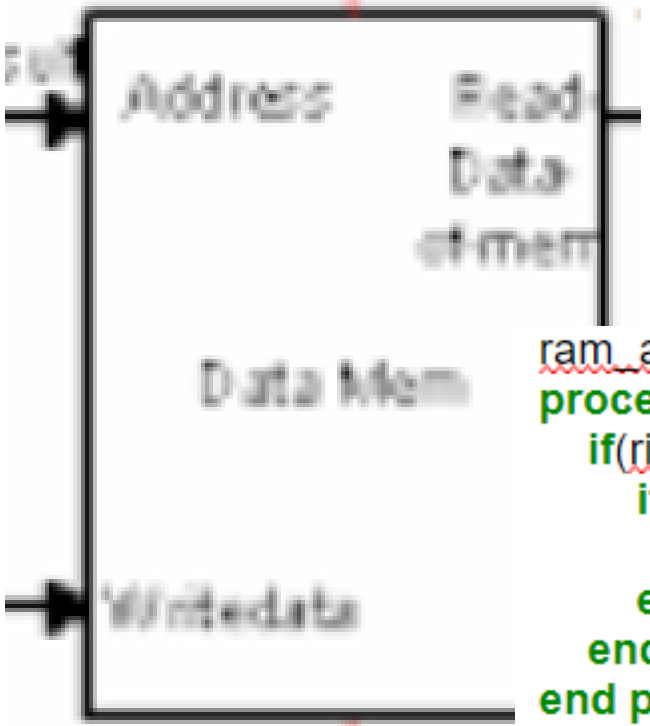

```
-- data memory of the MIPS Processor in VHDL
data_memory: entity work.Data_Memory_VHDL port map
  (
  clk => clk,
  mem_access_addr => ALU_out,
  mem_write_data => reg_read_data_2,
  mem_write_en => mem_write,
  mem_read => mem_read,
  mem_read_data => mem_read_data
  );
-- write back of the MIPS Processor in VHDL
 reg_write_data <= pc2 when (mem_to_reg = "10") else
       mem_read_data when (mem_to_reg = "01") else ALU_out;
```

```vhdl
entity Data_Memory_VHDL is
port (
 clk: in std_logic;
 mem_access_addr: in std_logic_Vector(15 downto 0);
 mem_write_data: in std_logic_Vector(15 downto 0);
 mem_write_en,mem_read:in std_logic;
 mem_read_data: out std_logic_Vector(15 downto 0)
);
end Data_Memory_VHDL;

architecture Behavioral of Data_Memory_VHDL is
signal i: integer;
signal ram_addr: std_logic_vector(7 downto 0);
type data_mem is array (0 to 255 ) of std_logic_vector (15 downto 0);
signal RAM: data_mem :=((others=> (others=>'0')));
begin

 ram_addr <= mem_access_addr(8 downto 1);
 process(clk)
 begin
  if(rising_edge(clk)) then
  if (mem_write_en='1') then
  ram(to_integer(unsigned(ram_addr))) <= mem_write_data;
  end if;
  end if;
 end process;
   mem_read_data <= ram(to_integer(unsigned(ram_addr))) when (mem_read='1') else :

end Behavioral;
```

Here too, as in case of Program Counter PC, the address is assumed to be of "byte" in the memory.

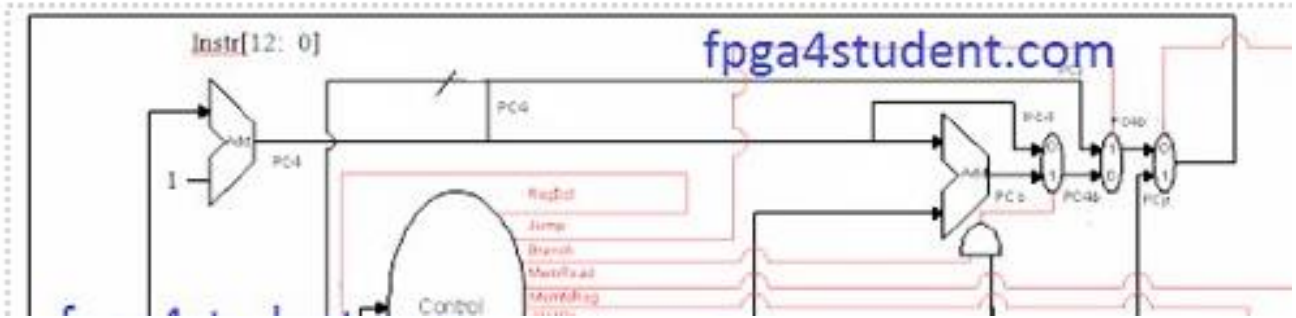
Since this memory is an array of 28 2-byte-wide datawords, we use bits 8 downto 1 of the address**

```
architecture Behavioral of Data_Memory_VHDL is
    signal i: integer;
    signal ram_addr: std_logic_vector(7 downto 0);
    type data_mem is array (0 to 255 ) of std_logic_vector (15 downto 0);
    signal RAM: data_mem :=((others=> (others=>'0')));
begin
    ram_addr <= mem_access_addr(8 downto 1);
    process(clk) begin
        if(rising_edge(clk)) then
            if (mem_write_en='1') then
                ram(to_integer(unsigned(ram_addr))) <= mem_write_data;
            end if;
        end if;
    end process;
    mem_read_data <= ram(to_integer(unsigned(ram_addr)))
                                    when (mem_read='1') else x"0000";
end Behavioral ;
```




```
ram_addr <= mem_access_addr(8 downto 1);
process(clk) begin
    if(rising_edge(clk)) then
        if (mem_write_en='1') then
            ram(to_integer(unsigned(ram_addr))) <= mem_write_data;
        end if;
    end if;
end process;
mem_read_data <= ram(to_integer(unsigned(ram_addr)))
                                    when (mem_read='1') else x"0000";
```

Branch/Jump Circuit



I'll skip this part (read from MIPS uArch chapter of Harris Harris textbook)

```
im_shift_1 <= imm_ext(14 downto 0) & '0';   -- immediate shift 1

no_sign_ext <= (not im_shift_1) + x"0001";

PC_beq <= (pc2 - no_sign_ext) when im_shift_1(15) = '1' else (pc2  + im_shift_1);

beq_control <= branch and zero_flag;    -- beq control

PC_4beq <= PC_beq when beq_control='1' else pc2;    -- PC_beq

jump_shift_1 <= instr(13 downto 0) & '0';   -- jump shift left 1

PC_j <= pc2(15) & jump_shift_1;    -- PC_j

PC_4beqj <= PC_j when jump = '1' else PC_4beq;    -- PC_4beqj

PC_jr <= reg_read_data_1;    -- PC_jr

pc_next <= PC_jr when (JRControl='1') else PC_4beqj;    -- PC_next
```

```
im_shift_1 <= imm_ext(14 downto 0) & '0';   -- immediate shift 1

no_sign_ext <= (not im_shift_1) + x"0001";

PC_beq <= (pc2 - no_sign_ext) when im_shift_1(15) = '1' else (pc2  + im_shift_1);

beq_control <= branch and zero_flag;     -- beq control

PC_4beq <= PC_beq when beq_control='1' else pc2;    -- PC_beq

jump_shift_1 <= instr(13 downto 0) & '0';   -- jump shift left 1

PC_j <= pc2(15) & jump_shift_1;     -- PC_j

PC_4beqj <= PC_j when jump = '1' else PC_4beq;    -- PC_4beqj

PC_jr <= reg_read_data_1;   -- PC_jr

pc_next <= PC_jr when (JRControl='1') else PC_4beqj;    -- PC_next
```

SKIPPED!

ALU related (left as exercise)

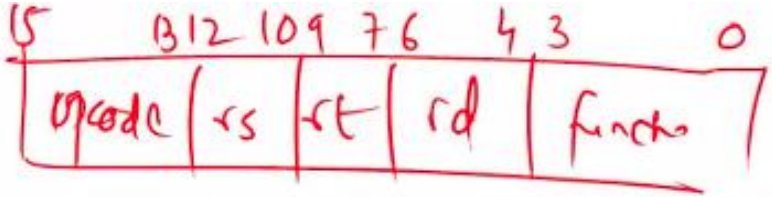
```
 pc2 <= pc_current + x"0002";
Instruction_Memory: entity work.Instruction_Memory_VHDL
        port map       (        pc=> pc_current ,       instruction => instr  );
control: entity work.control_unit_VHDL
   port map   (reset => reset,    opcode => instr(15 downto 13),
    reg_dst => reg_dst,    mem_to_reg => mem_to_reg,
    alu_op => alu_op,    jump => jump,    branch => branch,    mem_read => mem_read,
    mem_write => mem_write,    alu_src => alu_src,
    reg_write => reg_write,    sign_or_zero => sign_or_zero  );

 reg_write_dest <= "111" when  reg_dst= "10" else
        instr(6 downto 4) when  reg_dst= "01" else        instr(9 downto 7);
 reg_read_addr_1 <= instr(12 downto 10);   reg_read_addr_2 <= instr(9 downto 7);
 register_file: entity work.register_file_VHDL port map
                                ( clk => clk, rst => reset,  reg_write_en => reg_write,
 reg_write_dest => reg_write_dest,  reg_write_data => reg_write_data,
 reg_read_addr_1 => reg_read_addr_1,  reg_read_data_1 => reg_read_data_1,
 reg_read_addr_2 => reg_read_addr_2,  reg_read_data_2 => reg_read_data_2 );
 read_data2 <= imm_ext when alu_src='1' else reg_read_data_2;
alu:  entity work.ALU_VHDL port map  (
   a => reg_read_data_1,    b => read_data2,
   alu_control => ALU_Control,   alu_result => ALU_out, zero => zero_flag  );
```

ALU Control				
ALU op	**Function**	**ALUcnt**	**ALU Operation**	**Instruction**
11	**xxxx**	**000**	**ADD**	**Addi,lw,sw**
01	**xxxx**	**001**	**SUB**	**BEQ**
00	**00**	**000**	**ADD**	**R-type: ADD**
00	**01**	**001**	**SUB**	**R-type: sub**
00	**02**	**010**	**AND**	**R-type: AND**
00	**03**	**011**	**OR**	**R-type: OR**
00	**04**	**100**	**slt**	**R-type: slt**
10	**xxxxxx**	**100**	**slt**	**i-type: slti**

Already Discussed
in Previous LectureLabMeet-11

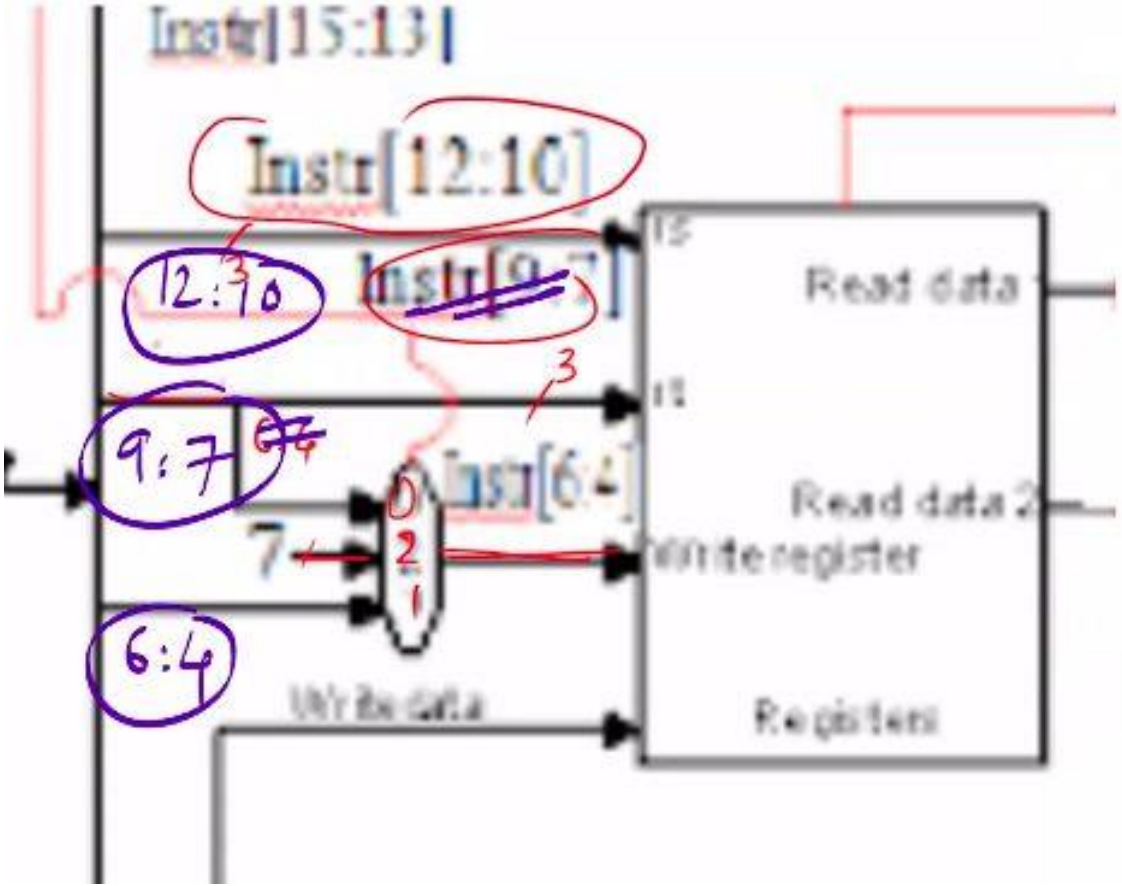
1. **Add : R[rd] = R[rs] + R[rt]**
2. **Subtract : R[rd] = R[rs] - R[rt]**
3. **And: R[rd] = R[rs] & R[rt]**
4. **Or : R[rd] = R[rs] | R[rt]**
5. **SLT: R[rd] = 1 if R[rs] < R[rt] else 0**
6. **Jr: PC=R[rs]**

RTN
register transfer notation



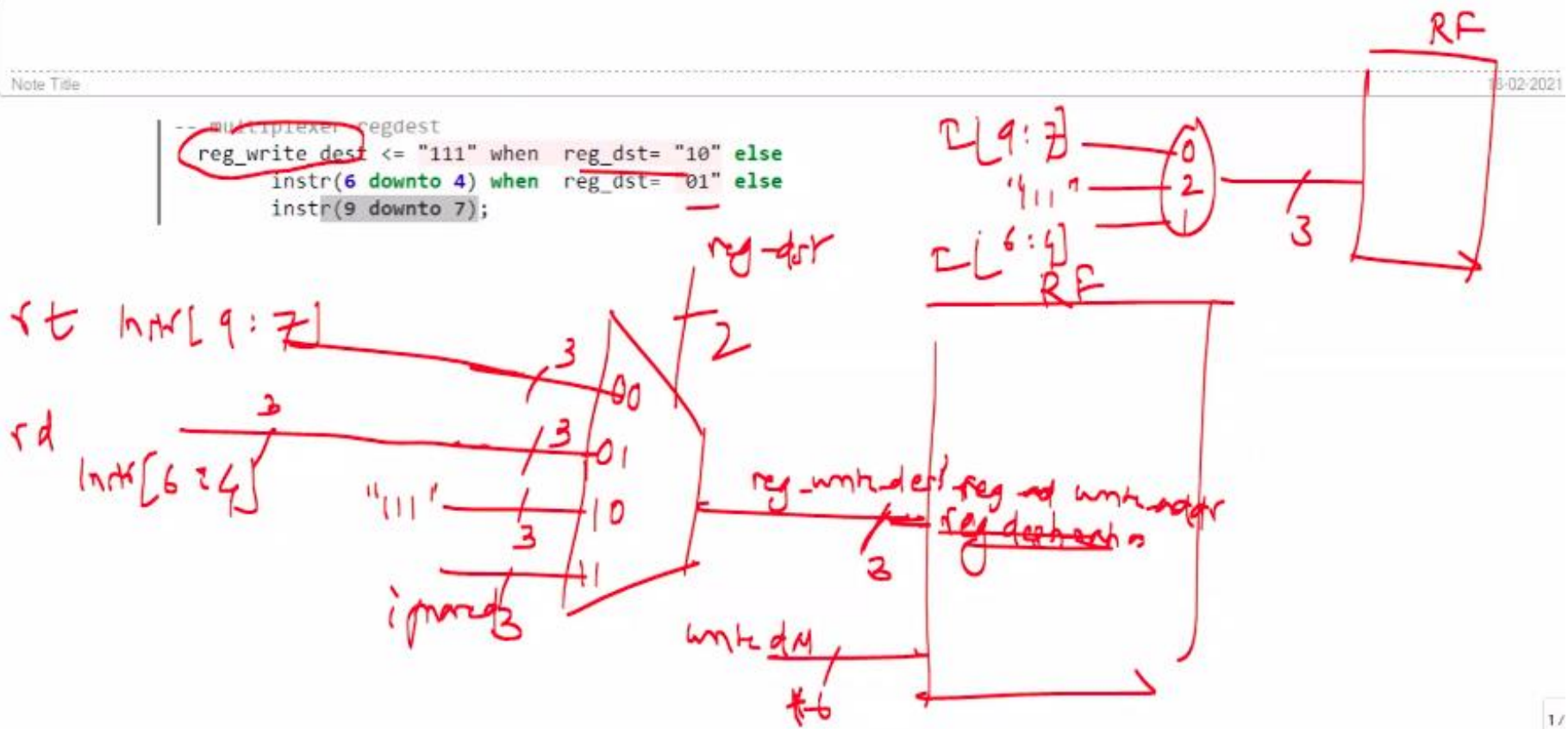
R-format

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	0	2	3	1	0	add $1,$2,$3
sub	R	0	2	3	1	1	sub $1,$2,$3
and	R	0	2	3	1	2	and $1,$2,$3
or	R	0	2	3	1	3	or $1,$2,$3
slt	R	0	2	3	1	4	slt $1,$2,$3
jr	R	0	7	0	0	8	jr $7
lw	I	4	2	1	7		lw $1, 7 ($2)
sw	I	5	2	1	7		sw $1, 7 ($2)
beq	I	6	1	2	7		beq $1,$2, 7
addi	I	7	2	1	7		addi $1,$2, 7
j	J	2	500				j 1000
jal	J	3	500				jal 1000
slti	I	1	2	1	7		slti $1,$2, 7


Instr[15:13]
Instr[12:10]
12:10
Instr[9:7]
9:7
Instr[6:4]
7
6:4
Read data
Read data 2
Write register
Registers

Note Title

```
-- multiplexer regdest
reg_write_dest <= "111" when  reg_dst= "10" else
        instr(6 downto 4) when  reg_dst= "01" else
        instr(9 downto 7);
```



5. SLT: R[rd] = 1 if R[rs] < R[rt] else 0
6. Jr: PC=R[rs]
7. Lw: R[rt] = M[R[rs]+SignExtImm]
8. Sw : M[R[rs]+SignExtImm] = R[rt]
9. Beq : if(R[rs]==R[rt]) PC=PC+1+BranchAddr
10. Addi: R[rt] = R[rs] + SignExtImm
11. J : PC=JumpAddr
12. Jal : R[7]=PC+2;PC=JumpAddr
13. SLTI: R[rt] = 1 if R[rs] < imm else 0

SignExtImm = { 9{immediate[6]}, imm

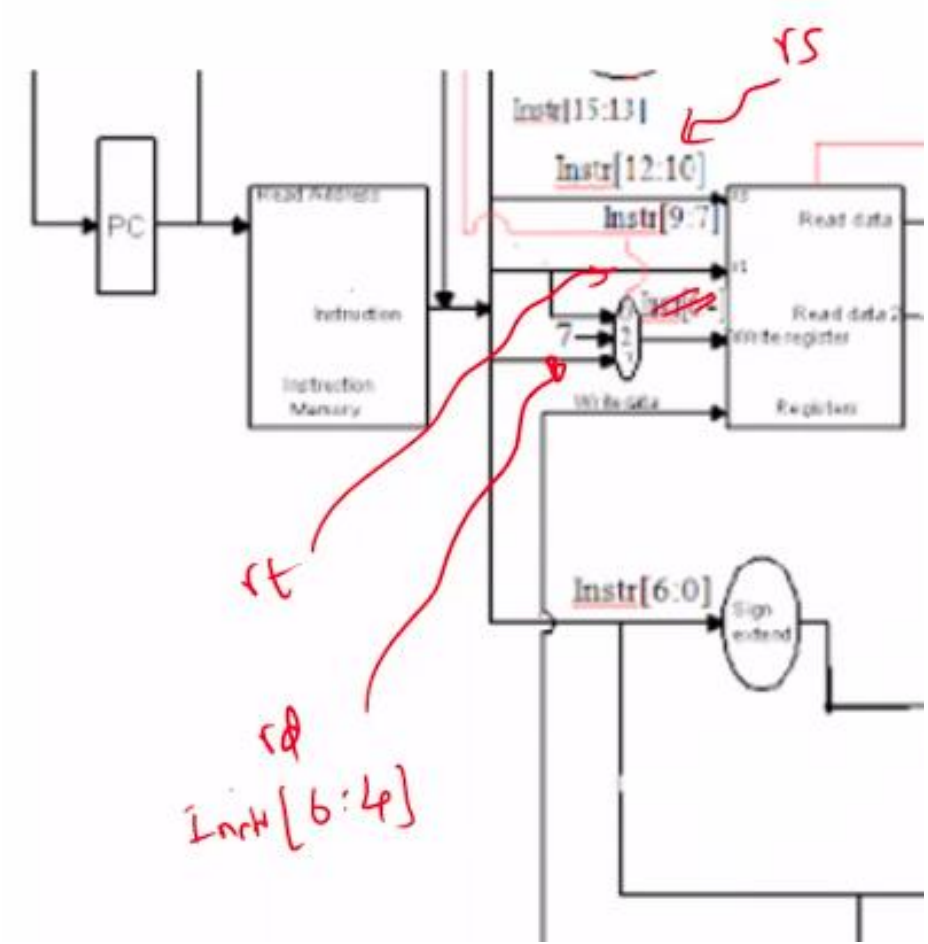
JumpAddr = { (PC+1)[15:13], address}

BranchAddr = { 7{immediate[6]}, immediate, 1'b0 }

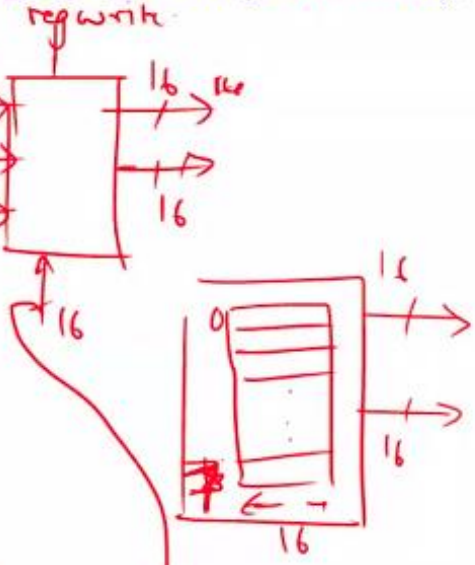
notation

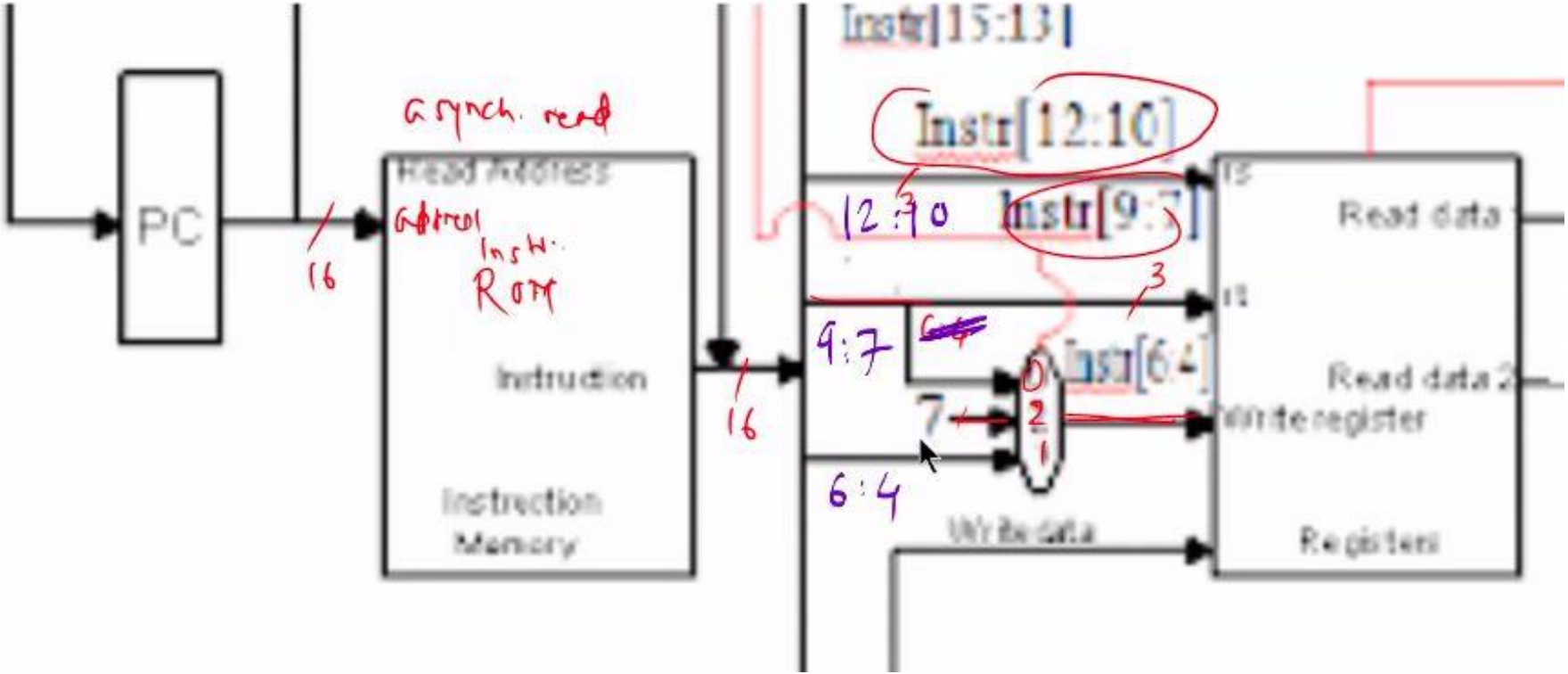
SLT:
set less than

Reg # 7 R[7] is used
for holding return address


rs
Instr[15:13]
Instr[12:10]
Instr[9:7]
PC
Instruction
Instruction Memory
Read data
Read data 2
Write register
Registers
7
rt
rd
Instr[6:0]
Sign extend
Instr[6:4]

```
entity register_file_VHDL is port (  clk,rst: in std_logic;  reg_write_en: in std_logic;
 reg_write_dest: in std_logic_vector(2 downto 0);
 reg_write_data: in std_logic_vector(15 downto 0);  reg_read_addr_1: in std_logic_vector(2 downto 0);
 reg_read_data_1: out std_logic_vector(15 downto 0);  reg_read_addr_2: in std_logic_vector(2 downto 0);
 reg_read_data_2: out std_logic_vector(15 downto 0)
);
end register_file_VHDL;
architecture Behavioral of register_file_VHDL is
    type reg_type is array (0 to 7 ) of std_logic_vector (15 downto 0);
    signal reg_array: reg_type;
begin
  process(clk,rst)  begin       if(rst='1') then  ..........................
      elsif(rising_edge(clk)) then
         if(reg_write_en='1') then
            reg_array(to_integer(unsigned(reg_write_dest))) <= reg_write_data;
         end if;
      end if;
  end process;
  reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else
                                                    reg_array(to_integer(unsigned(reg_read_addr_1)));
  reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else
                                                    reg_array(to_integer(unsigned(reg_read_addr_2)));
end Behavioral;
```

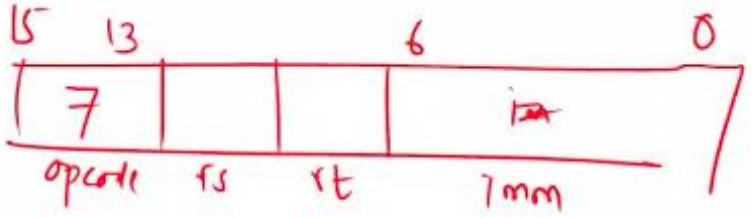



PC
16
Asynch. read
Instruction
Instruction Memory
16
Instr[15:13]
Instr[12:10]
12:10
Instr[9:7]
9:7
Instr[6:4]
6:4
0
1
2
7
Read data 1
Read data 2
Write register
Registers

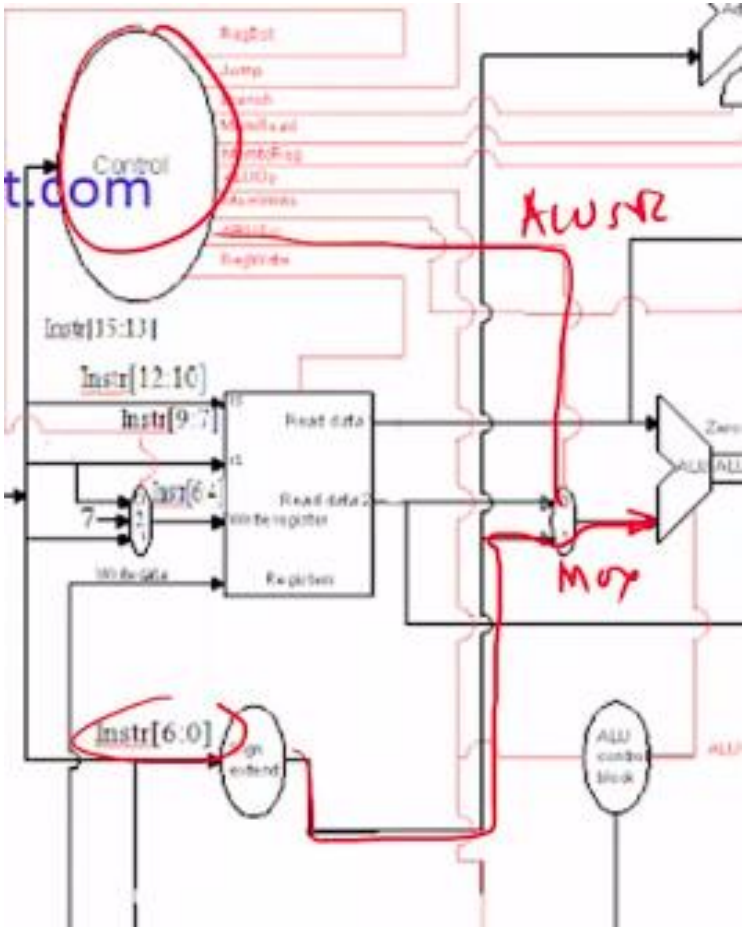
6. **Jr: PC=R[rs]**
7. **Lw: R[rt] = M[R[rs]+SignExtImm]**
8. **Sw : M[R[rs]+SignExtImm] = R[rt]**
9. **Beq : if(R[rs]==R[rt]) PC=PC+1+BranchAddr**
10. **Addi: R[rt] = R[rs] + SignExtImm**
11. **J : PC=JumpAddr**
12. **Jal : R[7]=PC+2;PC=JumpAddr**
13. **SLTI: R[rt] = 1 if R[rs] < imm else 0**

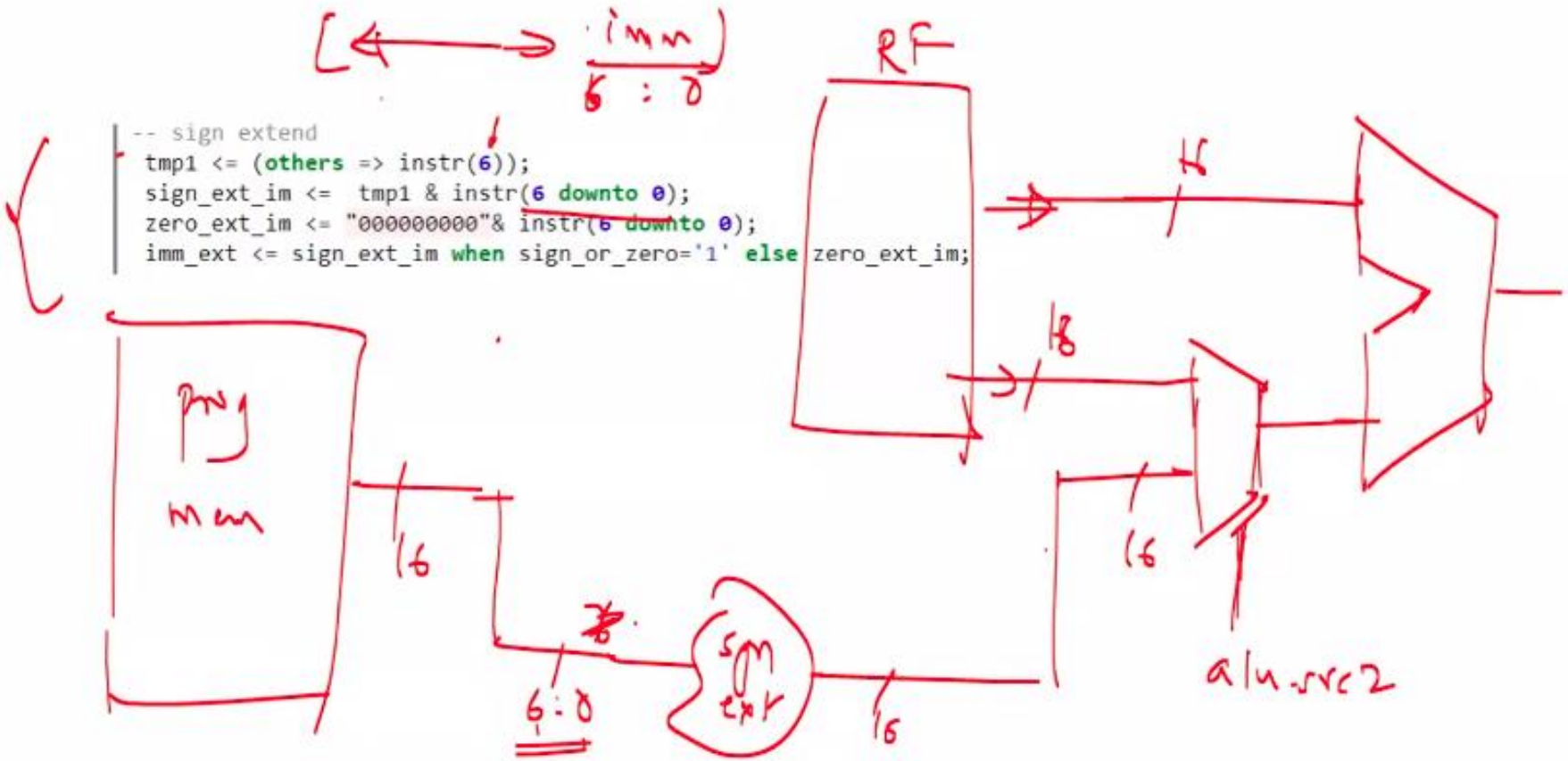
SignExtImm = { 9{immediate[6]}, imm

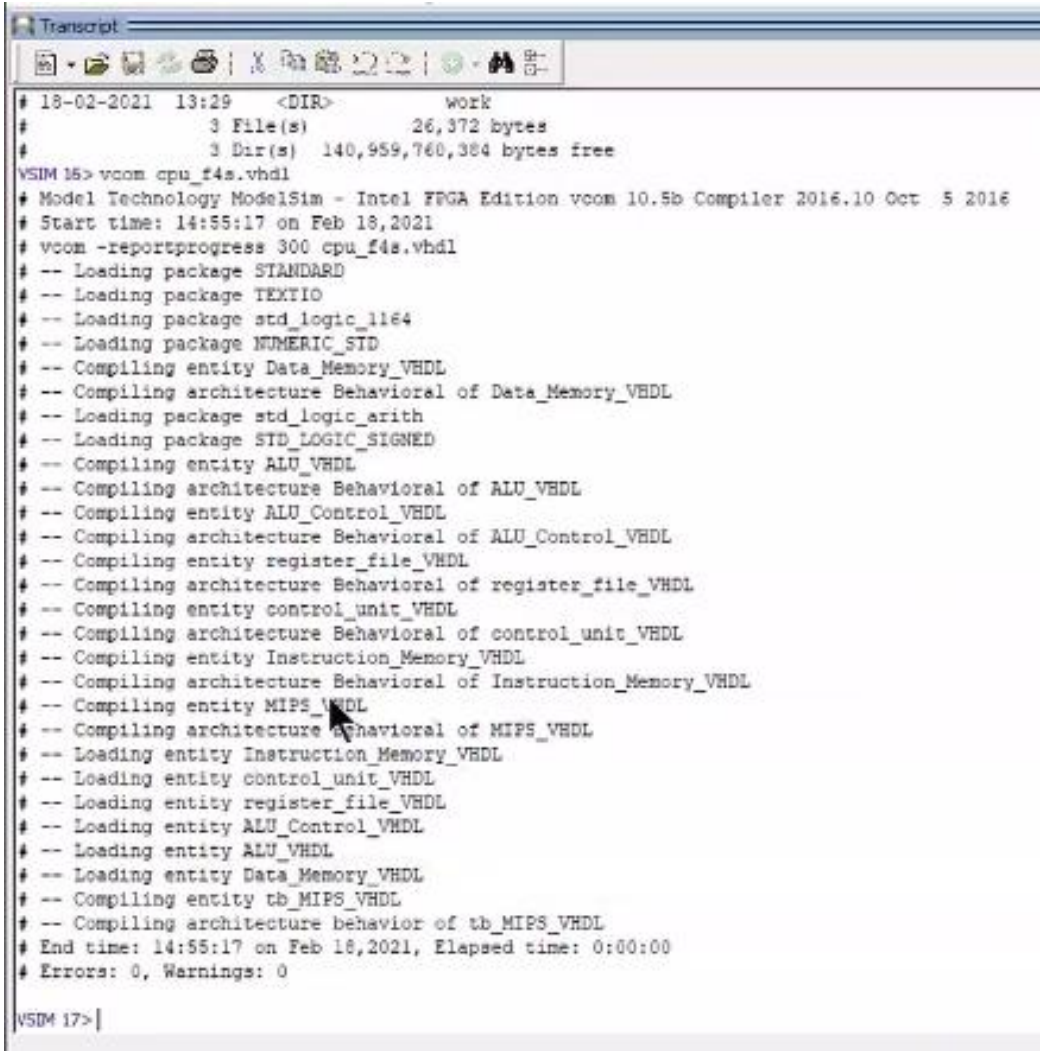
JumpAddr = { (PC+1)[15:13], address}



Sign Extended version of imm field


Control
ALU src
Instr[15:13]
Instr[12:10]
Instr[9:7]
Instr[6:4]
7
Instr[6:0]
Mem

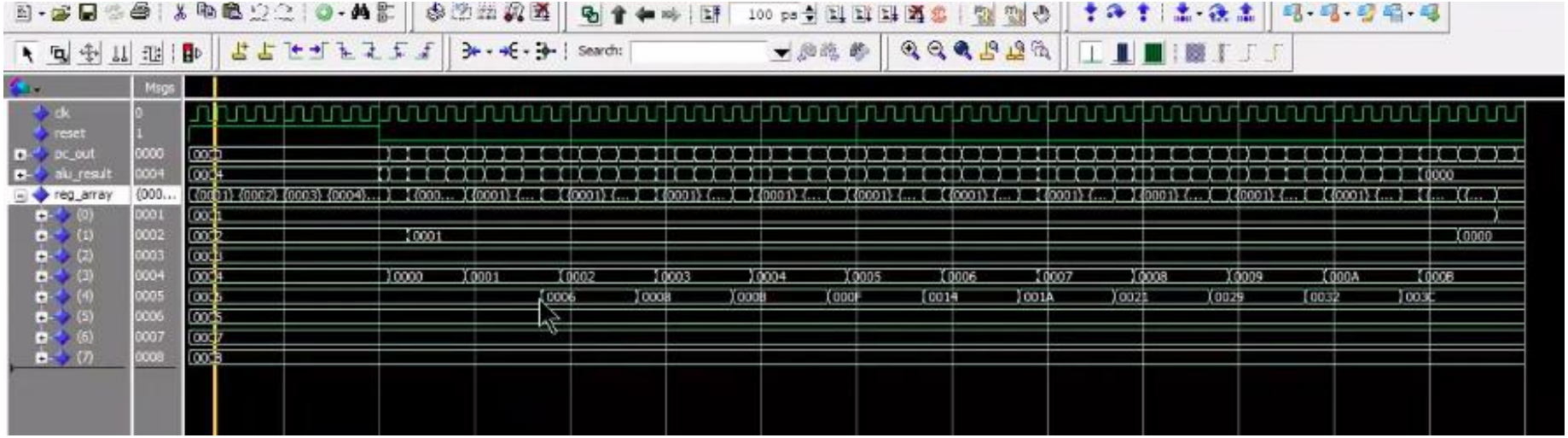

imm
6 : 0
RF
-- sign extend
tmp1 <= (others => instr(6));
sign_ext_im <= tmp1 & instr(6 downto 0);
zero_ext_im <= "000000000"& instr(6 downto 0);
imm_ext <= sign_ext_im when sign_or_zero='1' else zero_ext_im;
16
16
6
6 : 0
16
alu_src2

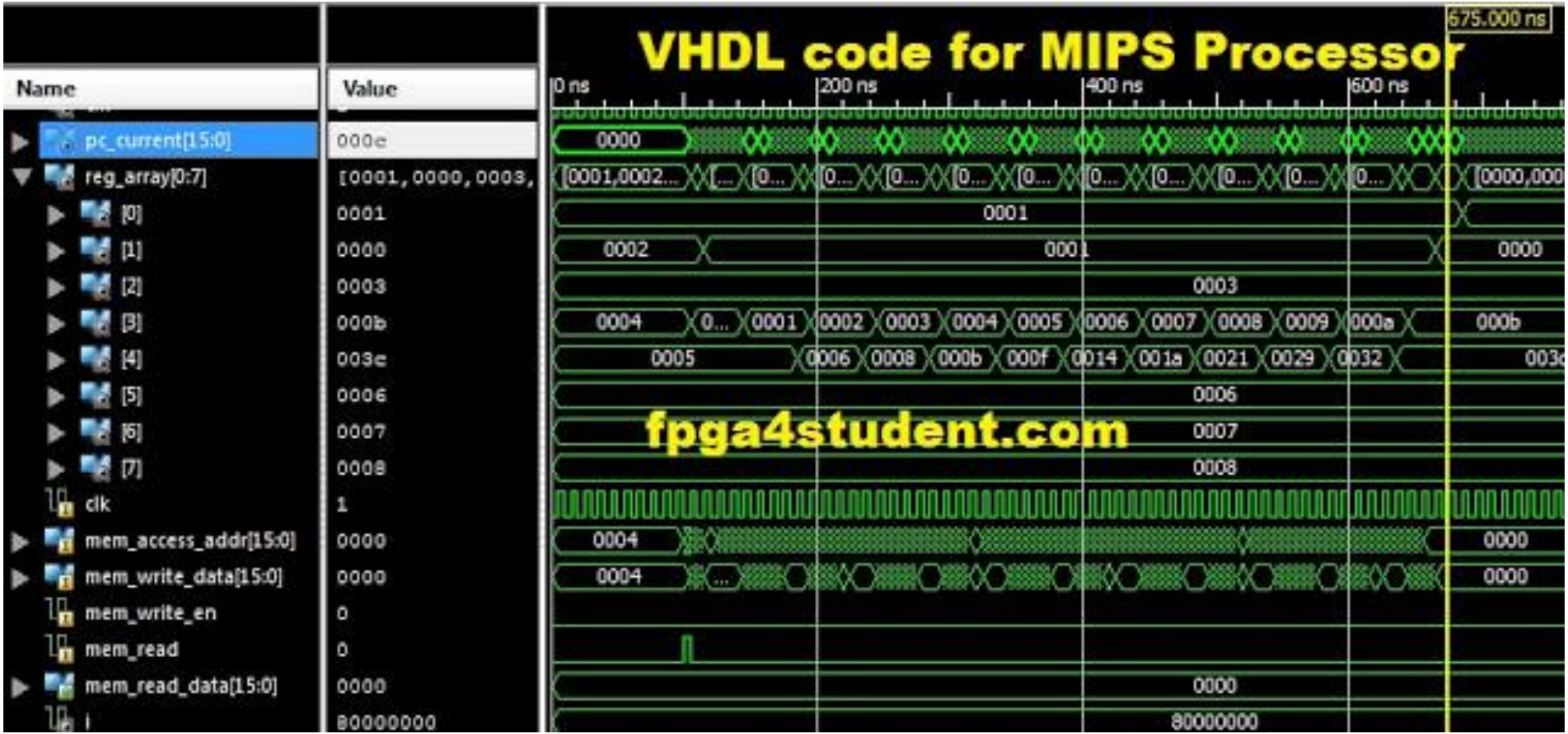

Transcript
18-02-2021 13:29 <DIR> work
3 File(s) 26,372 bytes
3 Dir(s) 140,959,760,384 bytes free
VSIM 16> vcom cpu_f4s.vhdl
Model Technology ModelSim - Intel FPGA Edition vcom 10.5b Compiler 2016.10 Oct 5 2016
Start time: 14:55:17 on Feb 18,2021
vcom -reportprogress 300 cpu_f4s.vhdl
-- Loading package STANDARD
-- Loading package TEXTIO
-- Loading package std_logic_1164
-- Loading package NUMERIC_STD
-- Compiling entity Data_Memory_VHDL
-- Compiling architecture Behavioral of Data_Memory_VHDL
-- Loading package std_logic_arith
-- Loading package STD_LOGIC_SIGNED
-- Compiling entity ALU_VHDL
-- Compiling architecture Behavioral of ALU_VHDL
-- Compiling entity ALU_Control_VHDL
-- Compiling architecture Behavioral of ALU_Control_VHDL
-- Compiling entity register_file_VHDL
-- Compiling architecture Behavioral of register_file_VHDL
-- Compiling entity control_unit_VHDL
-- Compiling architecture Behavioral of control_unit_VHDL
-- Compiling entity Instruction_Memory_VHDL
-- Compiling architecture Behavioral of Instruction_Memory_VHDL
-- Compiling entity MIPS_VHDL
-- Compiling architecture Behavioral of MIPS_VHDL
-- Loading entity Instruction_Memory_VHDL
-- Loading entity control_unit_VHDL
-- Loading entity register_file_VHDL
-- Loading entity ALU_Control_VHDL
-- Loading entity ALU_VHDL
-- Loading entity Data_Memory_VHDL
-- Compiling entity tb_MIPS_VHDL
-- Compiling architecture behavior of tb_MIPS_VHDL
End time: 14:55:17 on Feb 18,2021, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
VSIM 17>

File Edit View Bookmarks Window Help

Transcript

```
# -- Compiling entity register_file_VHDL
# -- Compiling architecture Behavioral of register_file_VHDL
# -- Compiling entity control_unit_VHDL
# -- Compiling architecture Behavioral of control_unit_VHDL
# -- Compiling entity Instruction_Memory_VHDL
# -- Compiling architecture Behavioral of Instruction_Memory_VHDL
# -- Compiling entity MIPS_VHDL
# -- Compiling architecture Behavioral of MIPS_VHDL
# -- Loading entity Instruction_Memory_VHDL
# -- Loading entity control_unit_VHDL
# -- Loading entity register_file_VHDL
# -- Loading entity ALU_Control_VHDL
# -- Loading entity ALU_VHDL
# -- Loading entity Data_Memory_VHDL
# -- Compiling entity tb_MIPS_VHDL
# -- Compiling architecture behavior of tb_MIPS_VHDL
# End time: 14:55:17 on Feb 18,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
VSIM 17> vsim  tb_MIPS_VHDL
# End time: 14:56:15 on Feb 18,2021, Elapsed time: 1:25:27
# Errors: 0, Warnings: 26
# vsim tb_MIPS_VHDL
# Start time: 14:56:15 on Feb 18,2021
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.tb_mips_vhdl(behavior)
# Loading ieee.std_logic_arith(body)
# Loading ieee.std_logic_signed(body)
# Loading ieee.numeric_std(body)
# Loading work.mips_vhdl(behavioral)
# Loading work.instruction_memory_vhdl(behavioral)
# Loading work.control_unit_vhdl(behavioral)
# Loading work.register_file_vhdl(behavioral)
# Loading work.alu_control_vhdl(behavioral)
# Loading work.alu_vhdl(behavioral)
# Loading work.data_memory_vhdl(behavioral)

VSIM 18>
```


Msgs
clk
reset
pc_out
alu_result
reg_array
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
Search:
100 ps
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
000A
000B
0006
0008
000F
0014
001A
0021
0029
0032
003C


VHDL code for MIPS Processor
fpga4student.com
Name
Value
pc_current[15:0]
reg_array[0:7]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
clk
mem_access_addr[15:0]
mem_write_data[15:0]
mem_write_en
mem_read
mem_read_data[15:0]
i
000e
[0001,0000,0003,
0001
0000
0003
000b
003e
0006
0007
0008
1
0000
0000
0
0
0000
80000000
0 ns
200 ns
400 ns
600 ns
675.000 ns