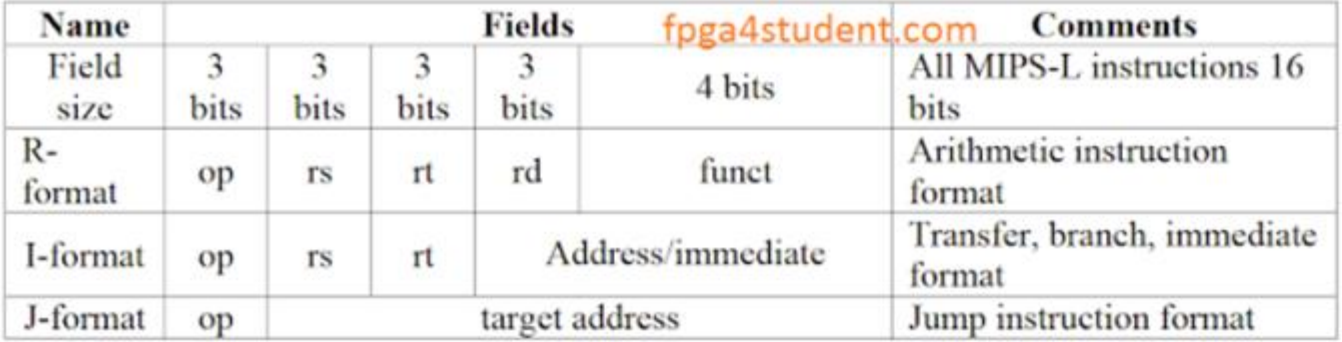
LecLabMeet-11-MIPS-vhdl

a walkthrough
part-1

Sachin B. Patkar

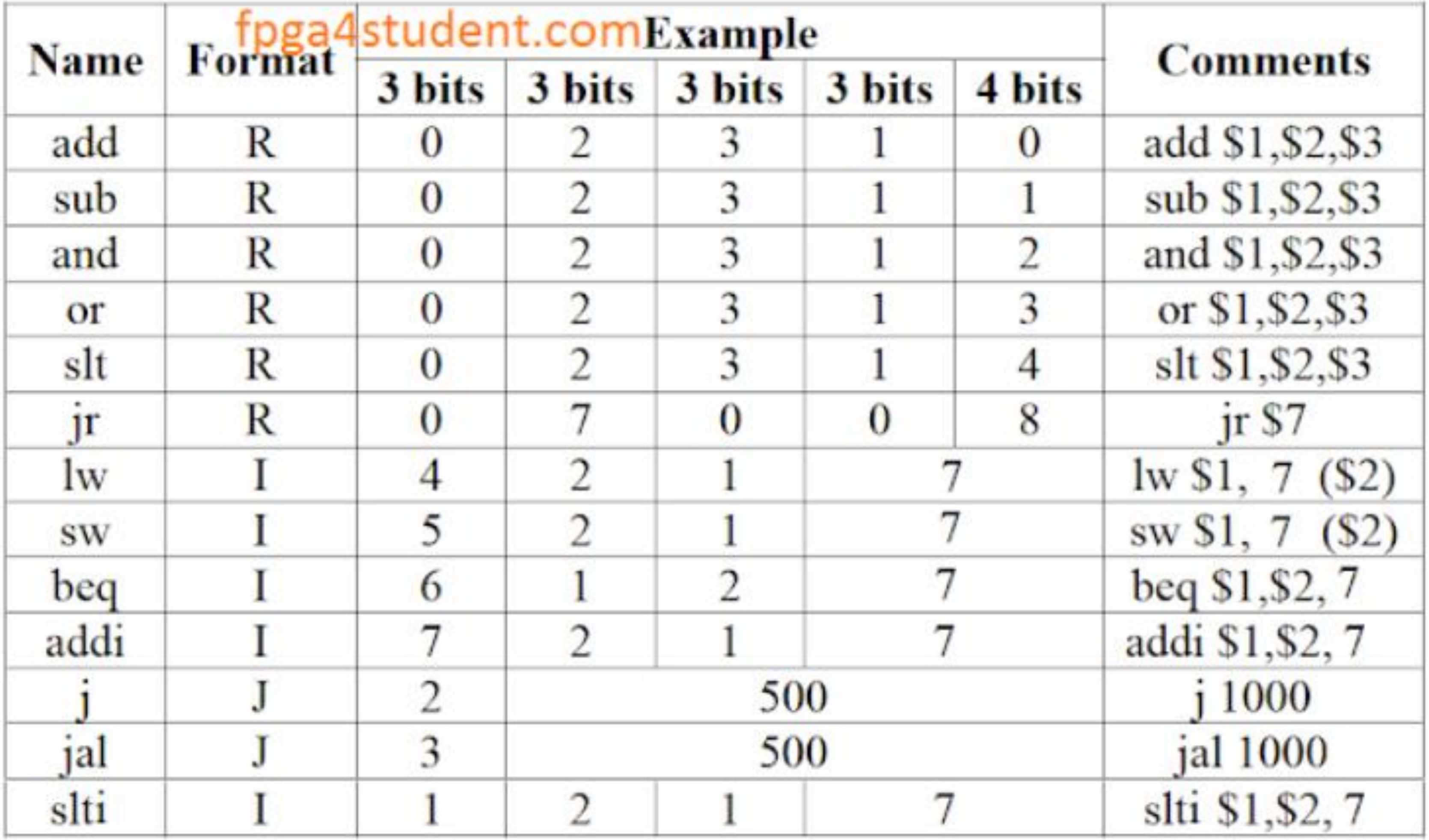
EE-705-spring-sem-2020-21 VLSI Design Lab
Thu-18-Feb-2021

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



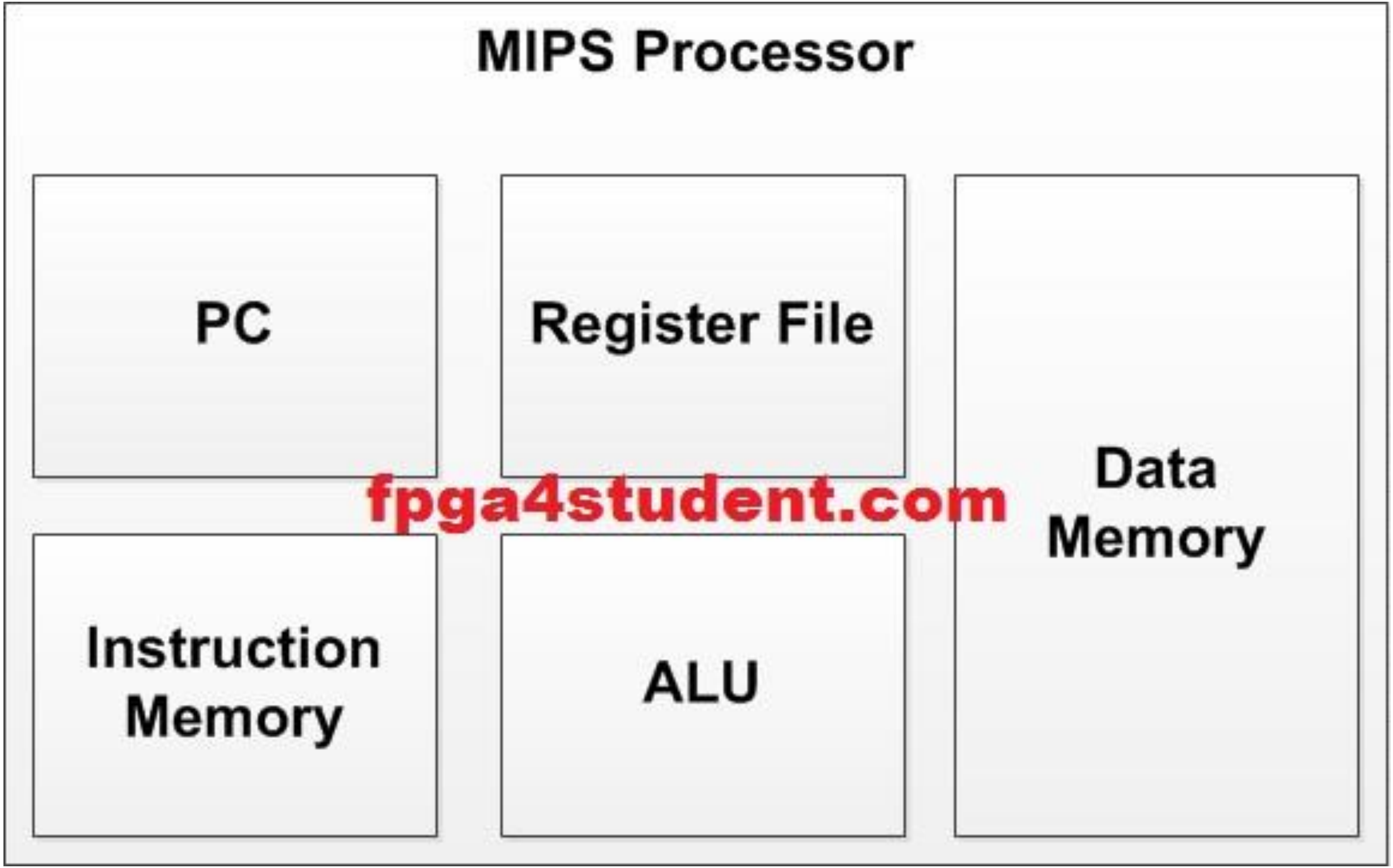
fpga4student.com

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



fpga4student.com

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



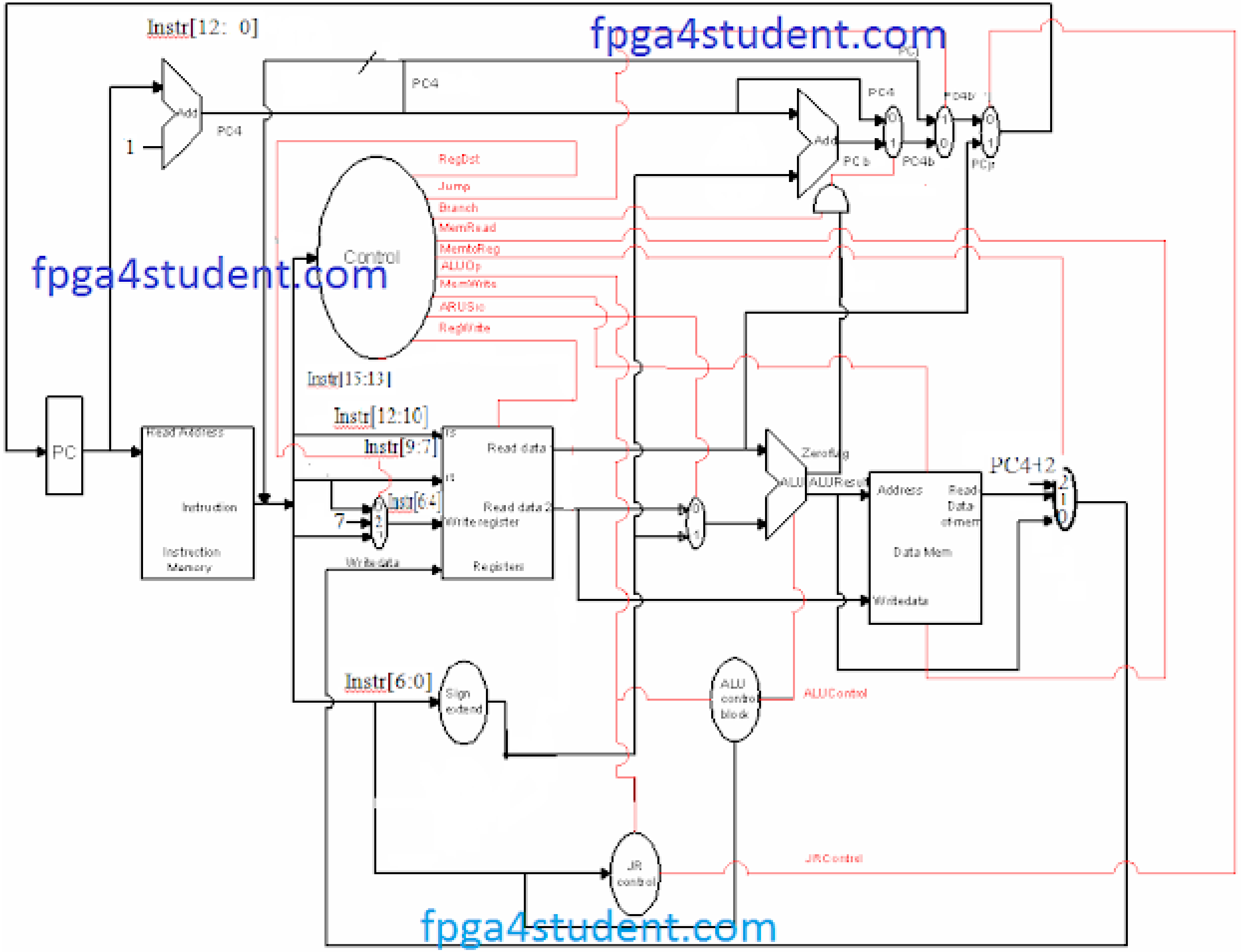
fpga4student.com

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

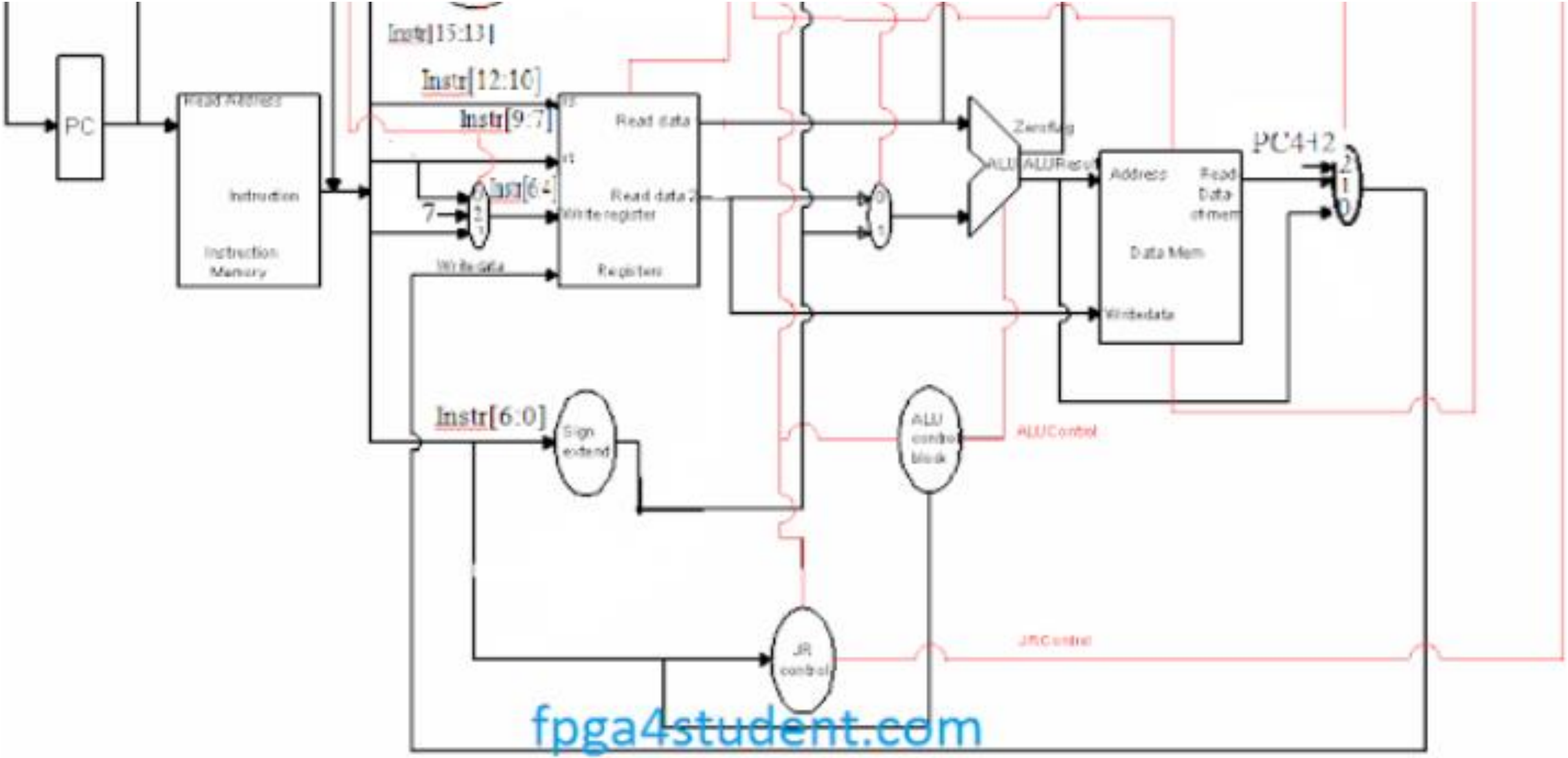
SignExtImm = { 9{immediate[6]}, imm

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

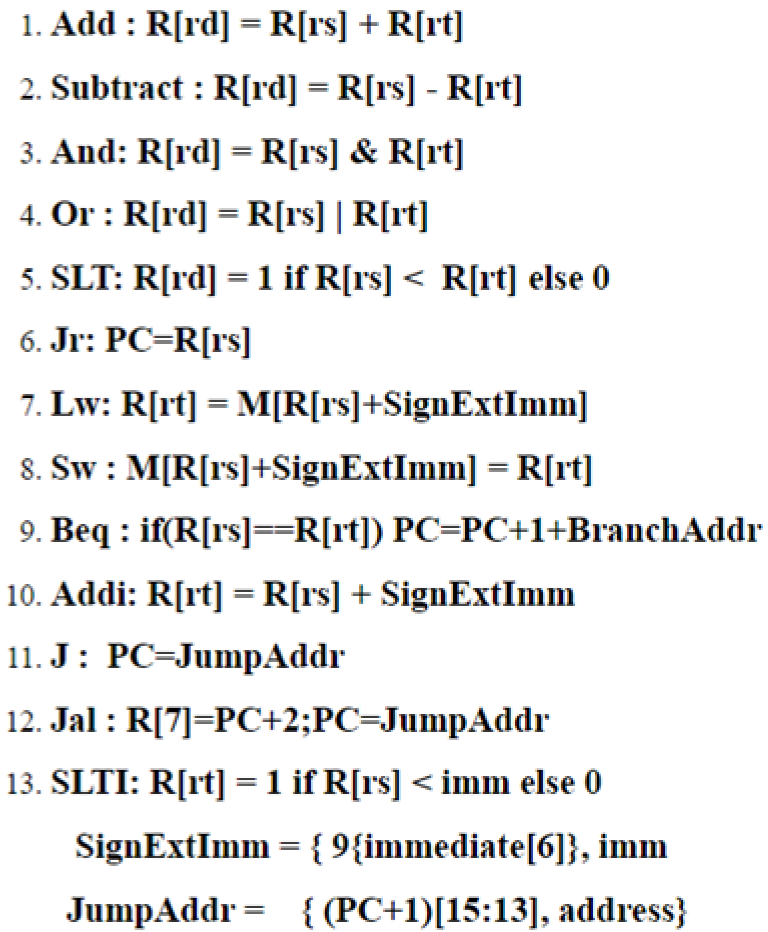
BranchAddr = { 7{immediate[6]}, immediate, 1’b0 }



fpga4student.com
fpga4student.com
fpga4student.com



fpga4student.com

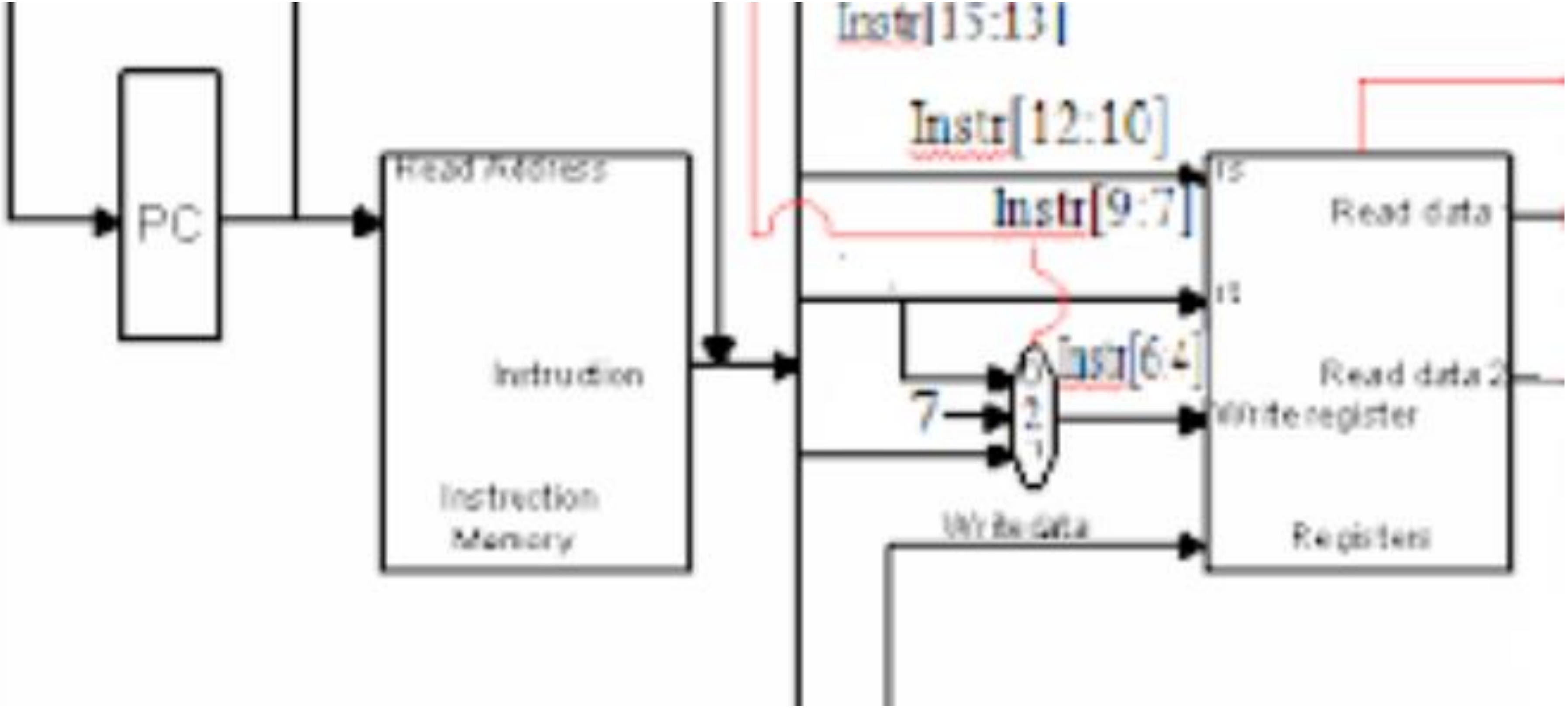


1. **Add : R[rd] = R[rs] + R[rt]**
2. **Subtract : R[rd] = R[rs] - R[rt]**
3. **And: R[rd] = R[rs] & R[rt]**
4. **Or : R[rd] = R[rs] | R[rt]**
5. **SLT: R[rd] = 1 if R[rs] < R[rt] else 0**
6. **Jr: PC=R[rs]**
7. **Lw: R[rt] = M[R[rs]+SignExtImm]**
8. **Sw : M[R[rs]+SignExtImm] = R[rt]**
9. **Beq : if(R[rs]==R[rt]) 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

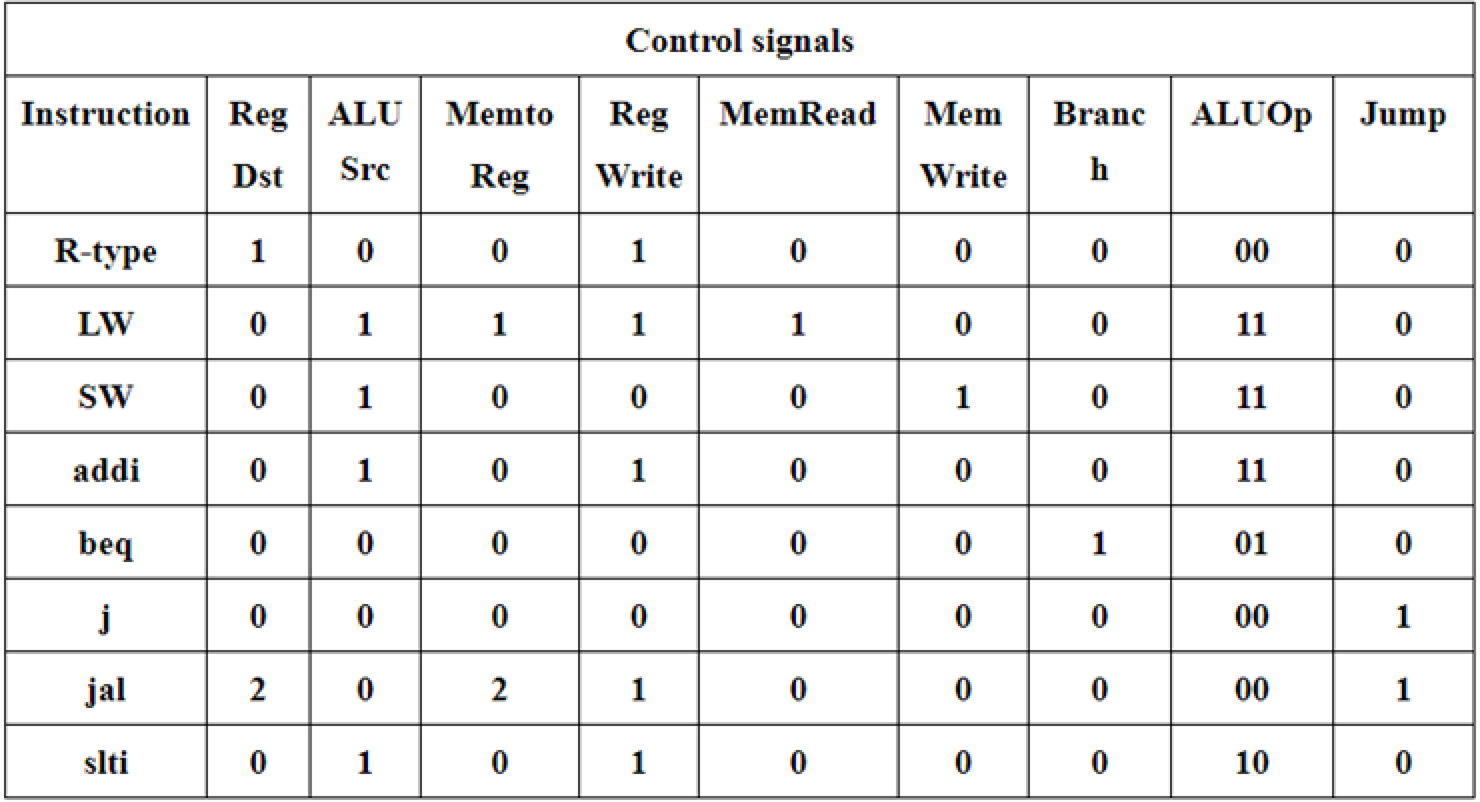
JumpAddr = { (PC+1)[15:13], address}

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

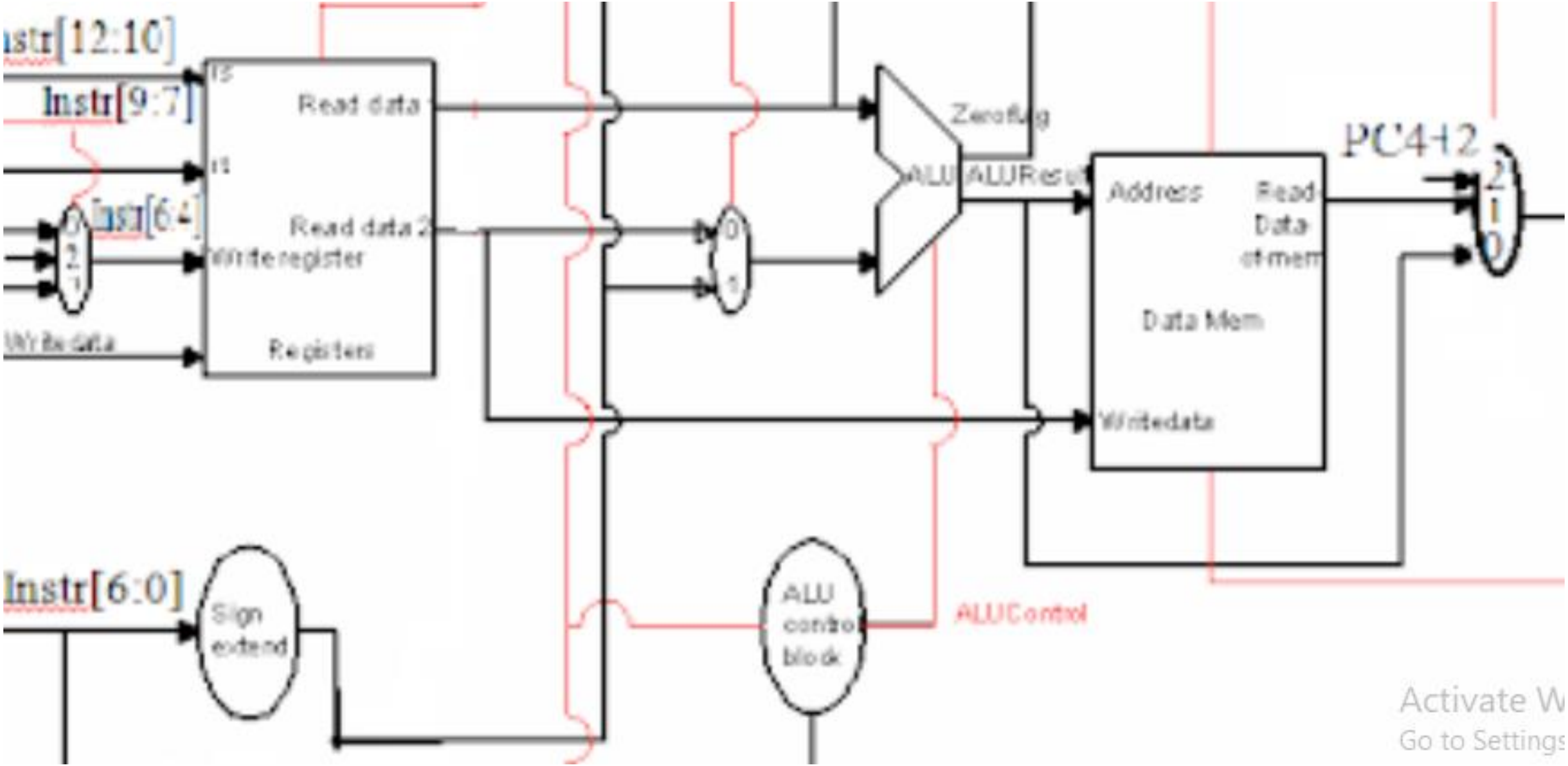

Instr[15:13]
Instr[12:10]
Instr[9:7]
Instr[6:4]
PC
Read Address
Instruction
Instruction Memory
rs
rt
Read data 1
Read data 2
Write register
Write data
Registers
0
2
1
7

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



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	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	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

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

```
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;
```

```vhdl
 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  );
```

```
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
--   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;
```

```vhdl
-- 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";
```

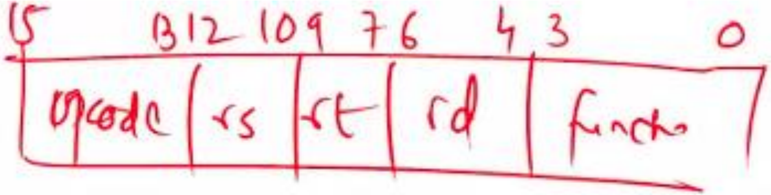
```
im_shift_1 <= imm_ext(14 downto 0) & '0';   -- immediate shift 1

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
```

```vhdl
-- 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;
```

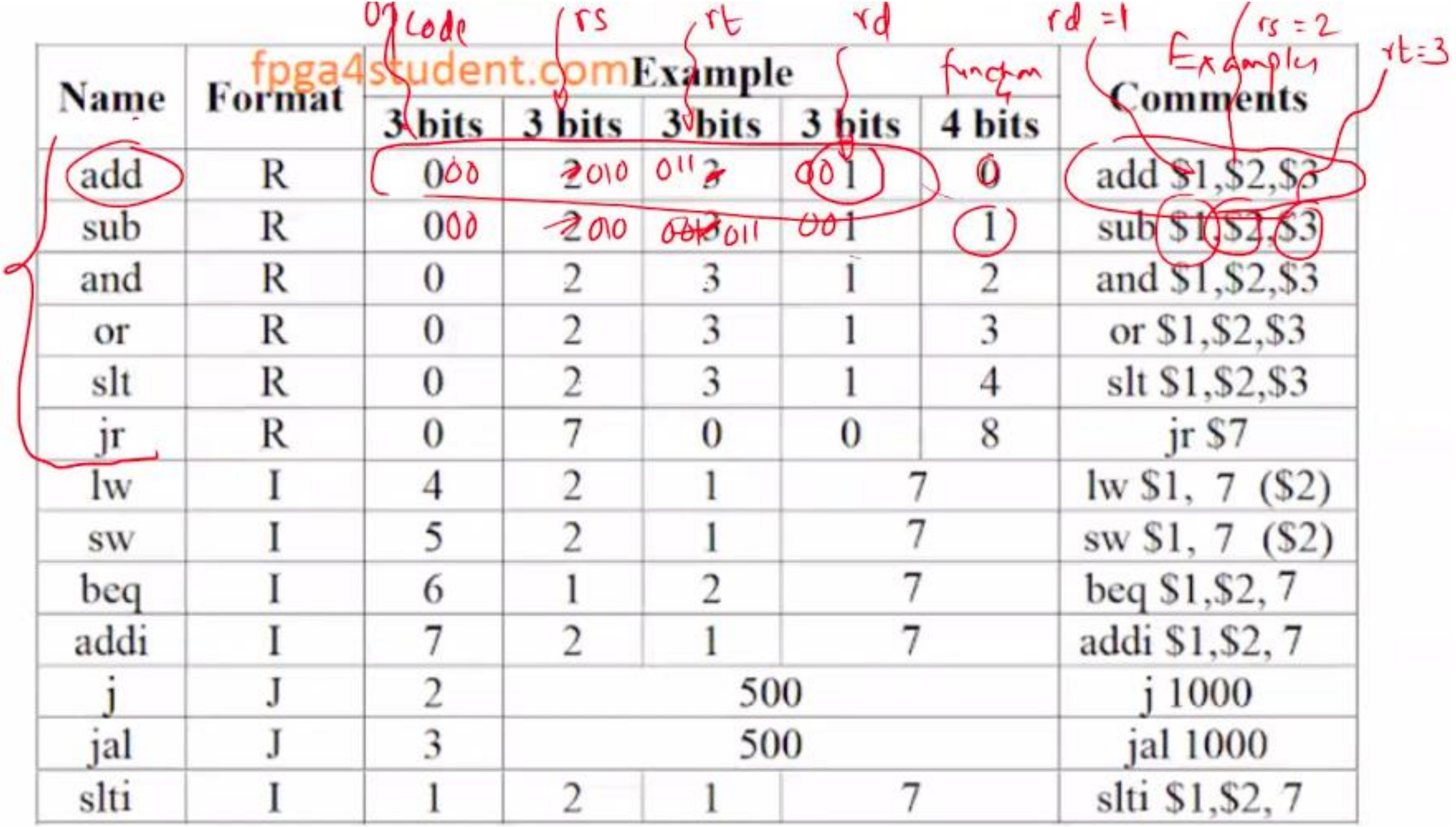
1. **Add : R[rd] = R[rs] + R[rt]**
2. **Subtract : R[rd] = R[rs] - R[rt]**
3. **And: R[rd] = R[rs] & R[rt]**
4. **Or : R[rd] = R[rs] | R[rt]**
5. **SLT: R[rd] = 1 if R[rs] < R[rt] else 0**
6. **Jr: PC=R[rs]**

RTN

register transfer notation



R-format

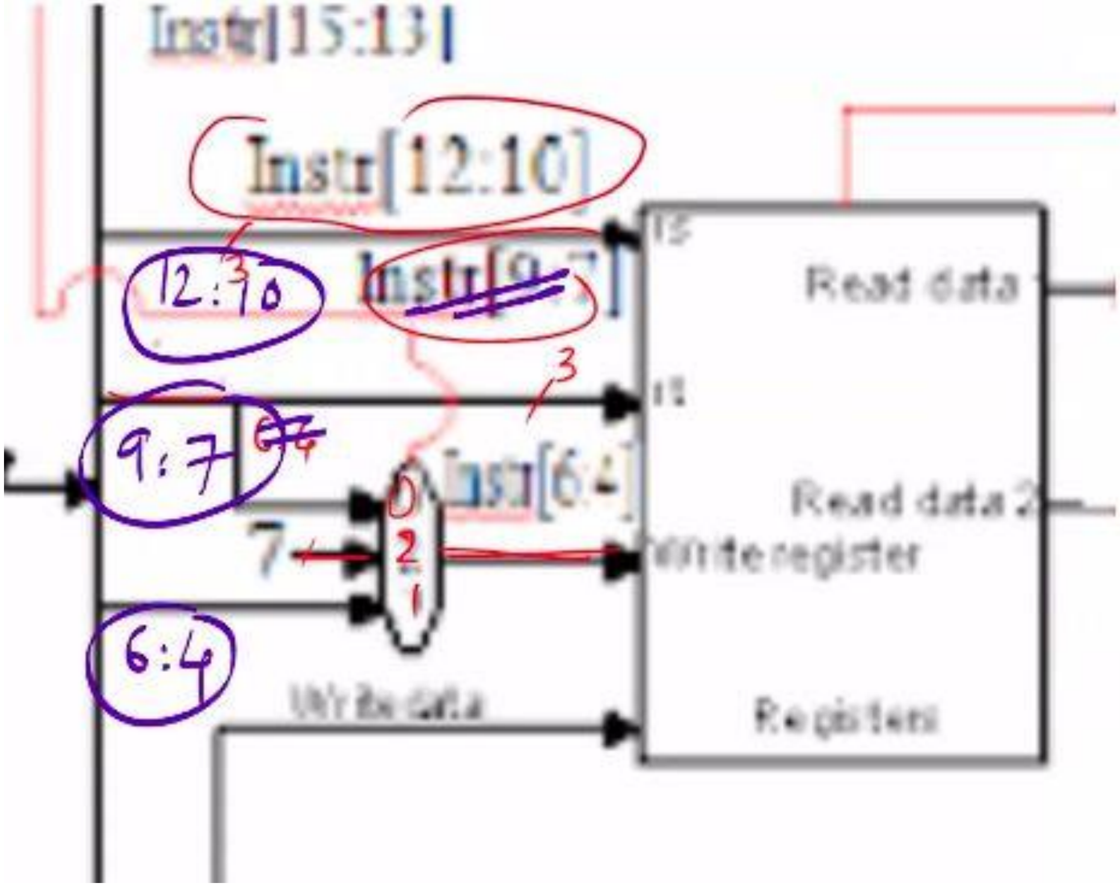


fpga4student.com

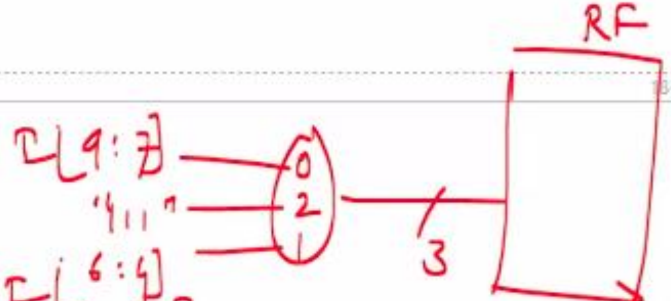
opcode rs rt rd function 4 rd =1 rs = 2 rt=3

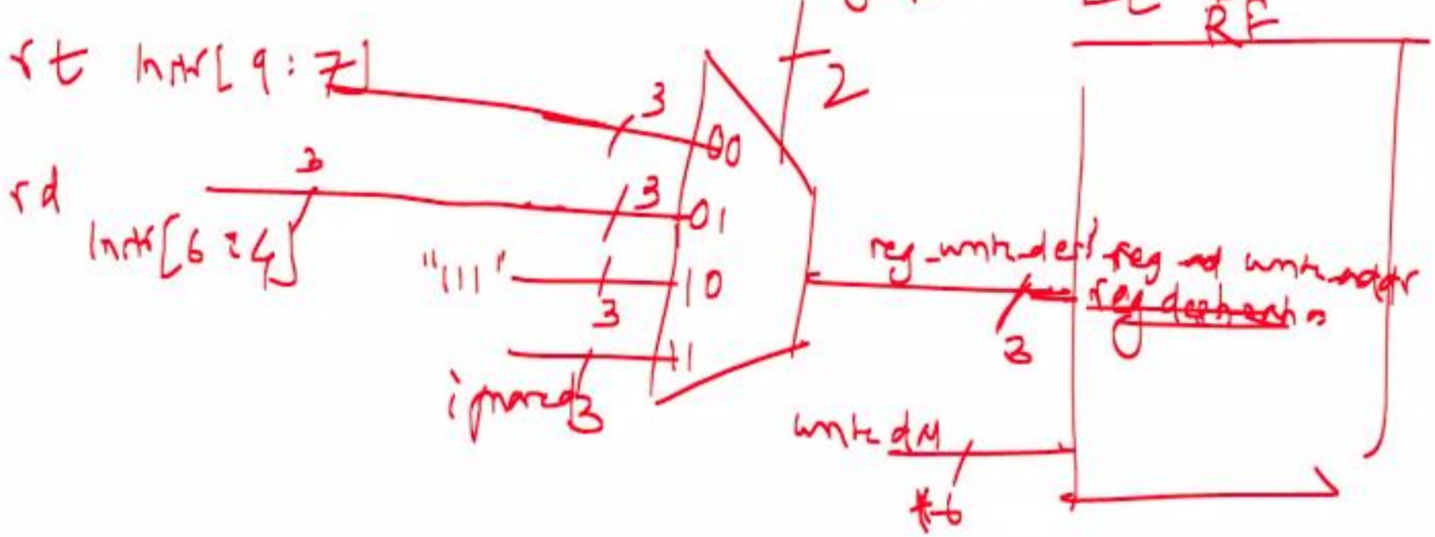
Example

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	000	~~2~~010	011~~3~~	001	0	add $1,$2,$3
sub	R	000	~~2~~010	~~0013~~011	001	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]
Instr[9:7]
Instr[6:4]
12:10
9:7
6:4
3
3
7
0
2
1
rs
rt
Write register
Write data
Read data 1
Read data 2
Registers

```
-- 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}

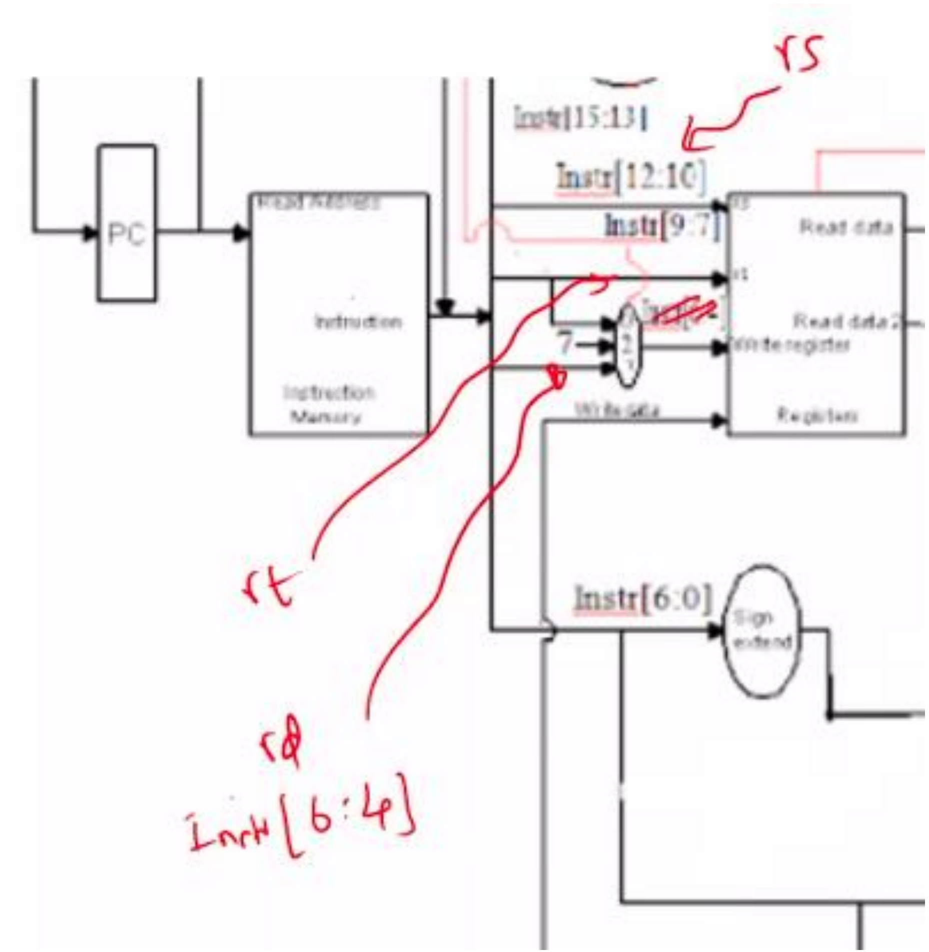
BranchAddr = { 7{immediate[6]}, immediate, 1'b0 }

notation

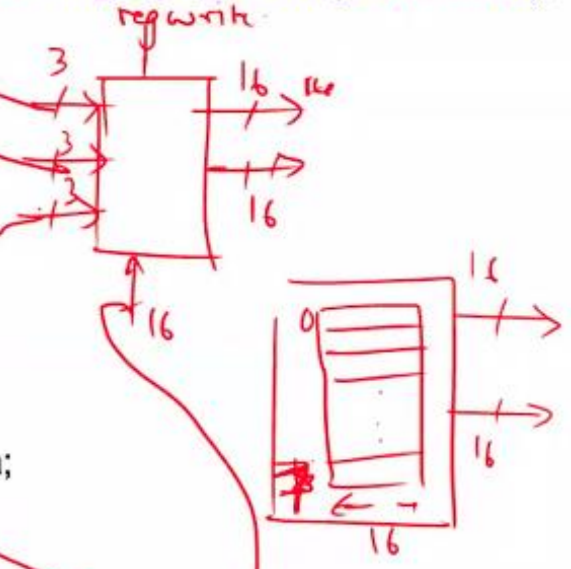
SLT:

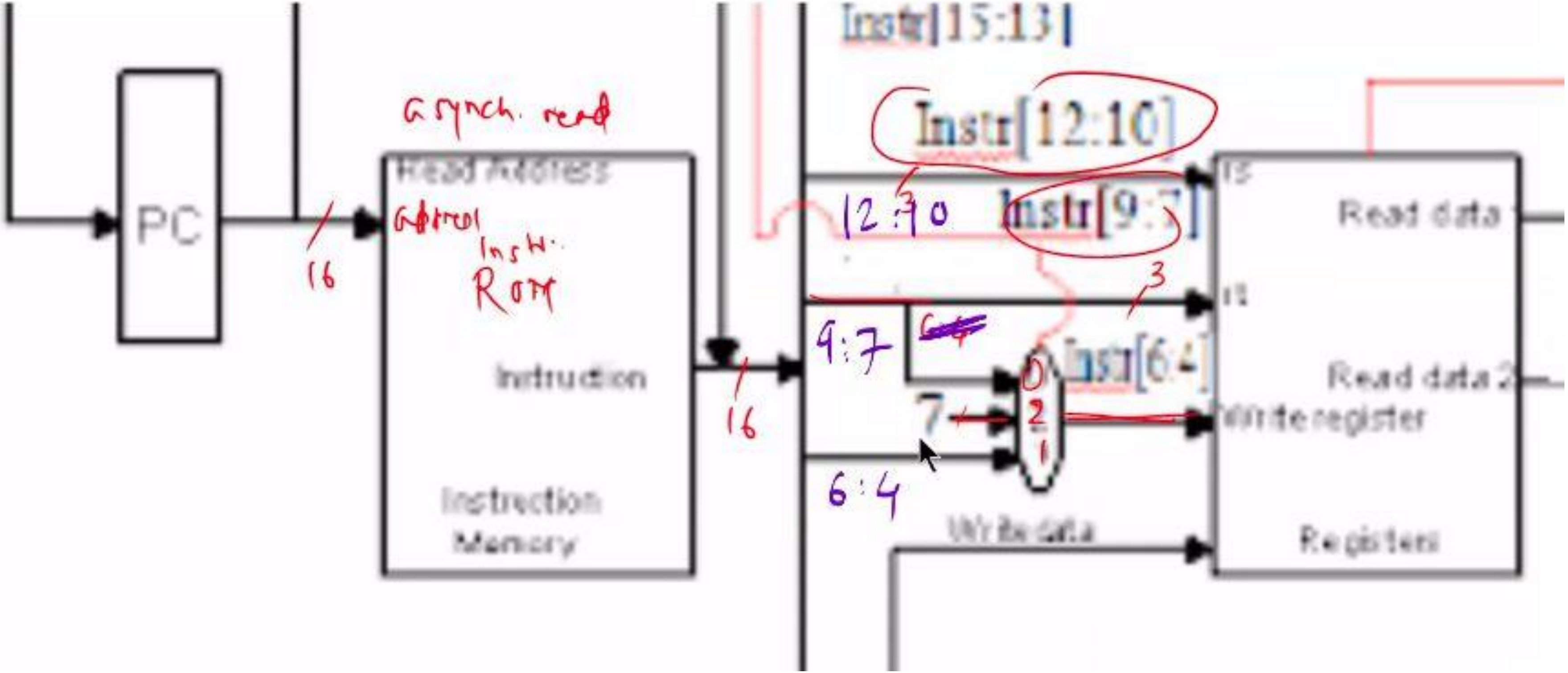
set less than

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


rs
Instr[15:13]
Instr[12:10]
Instr[9:7]
PC
Read Address
Instruction
Instruction Memory
rs
rt
Read data 1
Read data 2
7
Write register
Write data
Registers
Instr[6:0]
Sign extend
rt
rd
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;
```



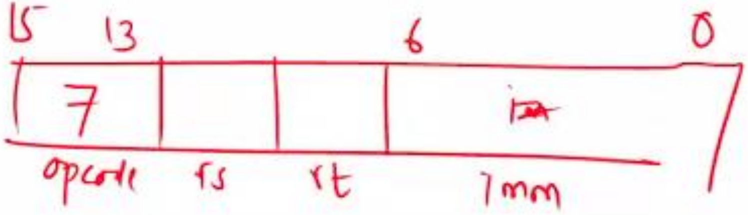


PC
16
asynch. read
Read Address
address
Instr.
ROM
Instruction
Instruction Memory
16
Instr[15:13]
Instr[12:10]
12:10
Instr[9:7]
9:7
3
Instr[6:4]
6:4
7
0
2
1
rs
rt
Write register
Write data
Read data 1
Read data 2
Registers

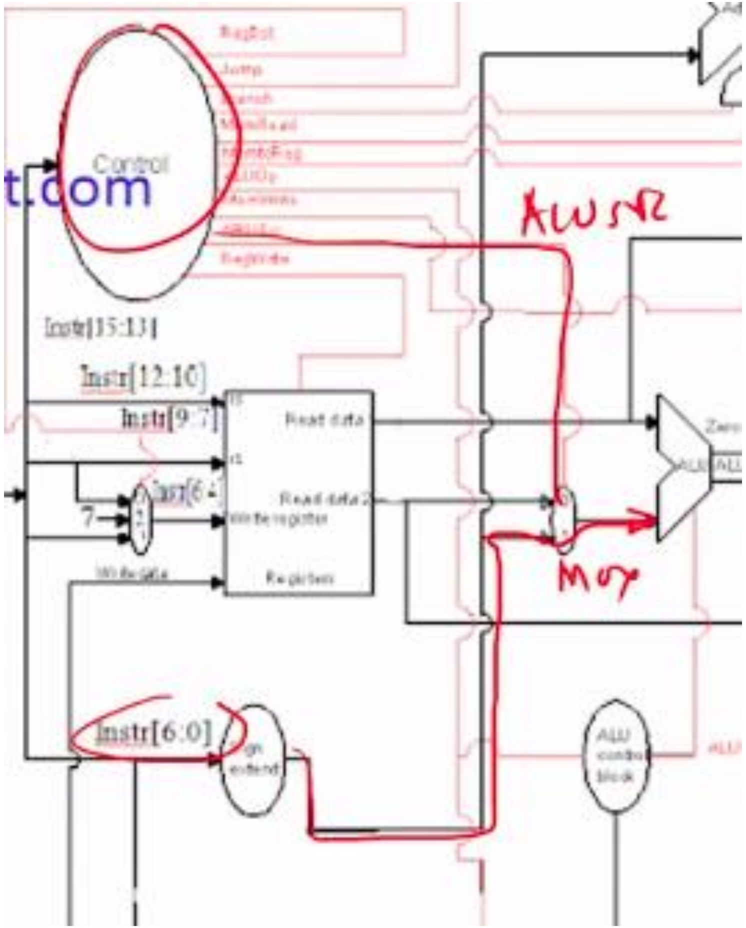
6. **Jr: PC=R[rs]**
7. **Lw: R[rt] = M[R[rs]+SignExtImm]**
8. **Sw : M[R[rs]+SignExtImm] = R[rt]**
9. **Beq : if(R[rs]==R[rt]) PC=PC+1+BranchAddr**
10. **Addi: R[rt] = R[rs] + SignExtImm**
11. **J : PC=JumpAddr**
12. **Jal : R[7]=PC+2;PC=JumpAddr**
13. **SLTI: R[rt] = 1 if R[rs] < imm else 0**

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

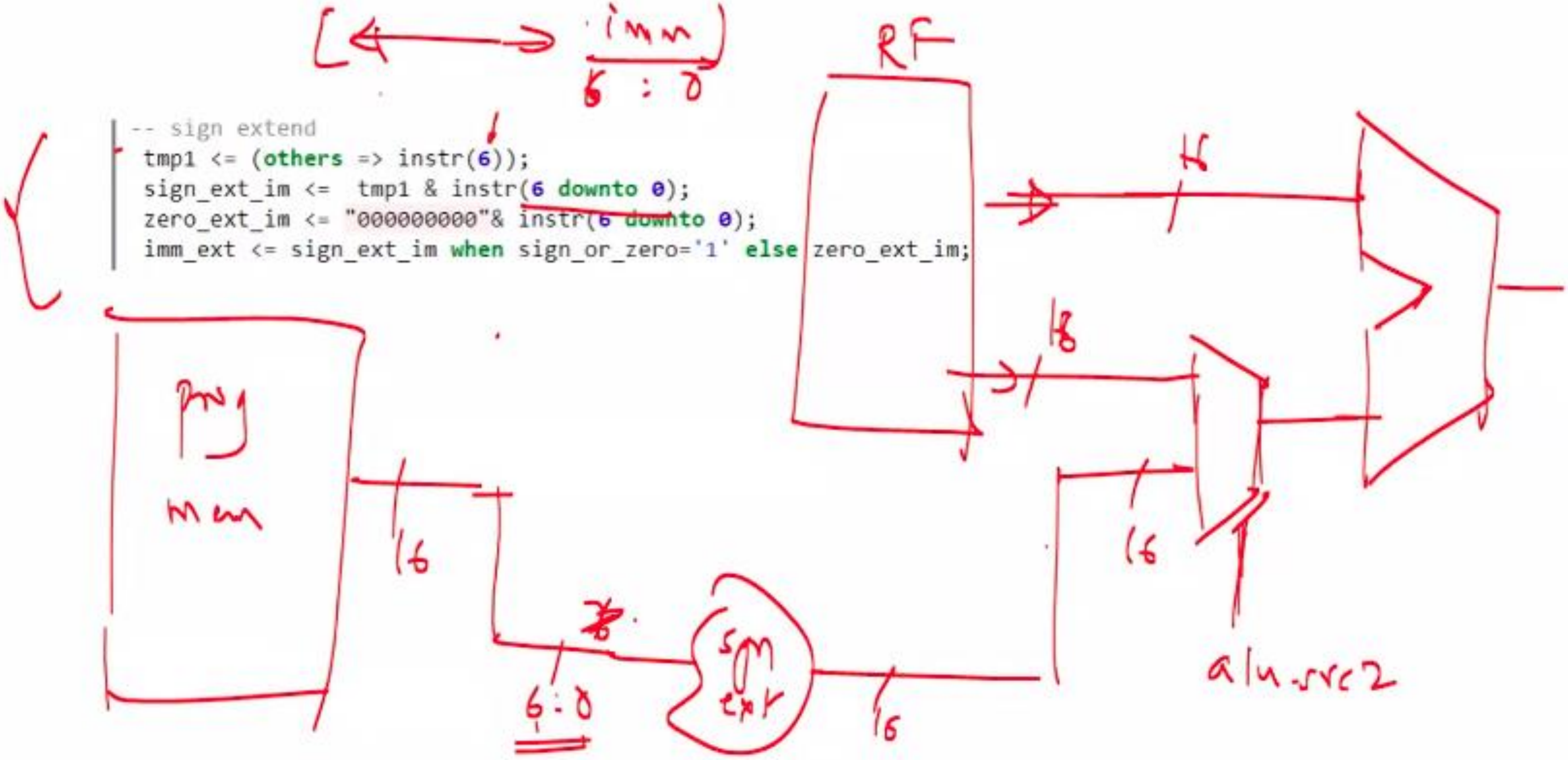
JumpAddr = { (PC+1)[15:13], address}



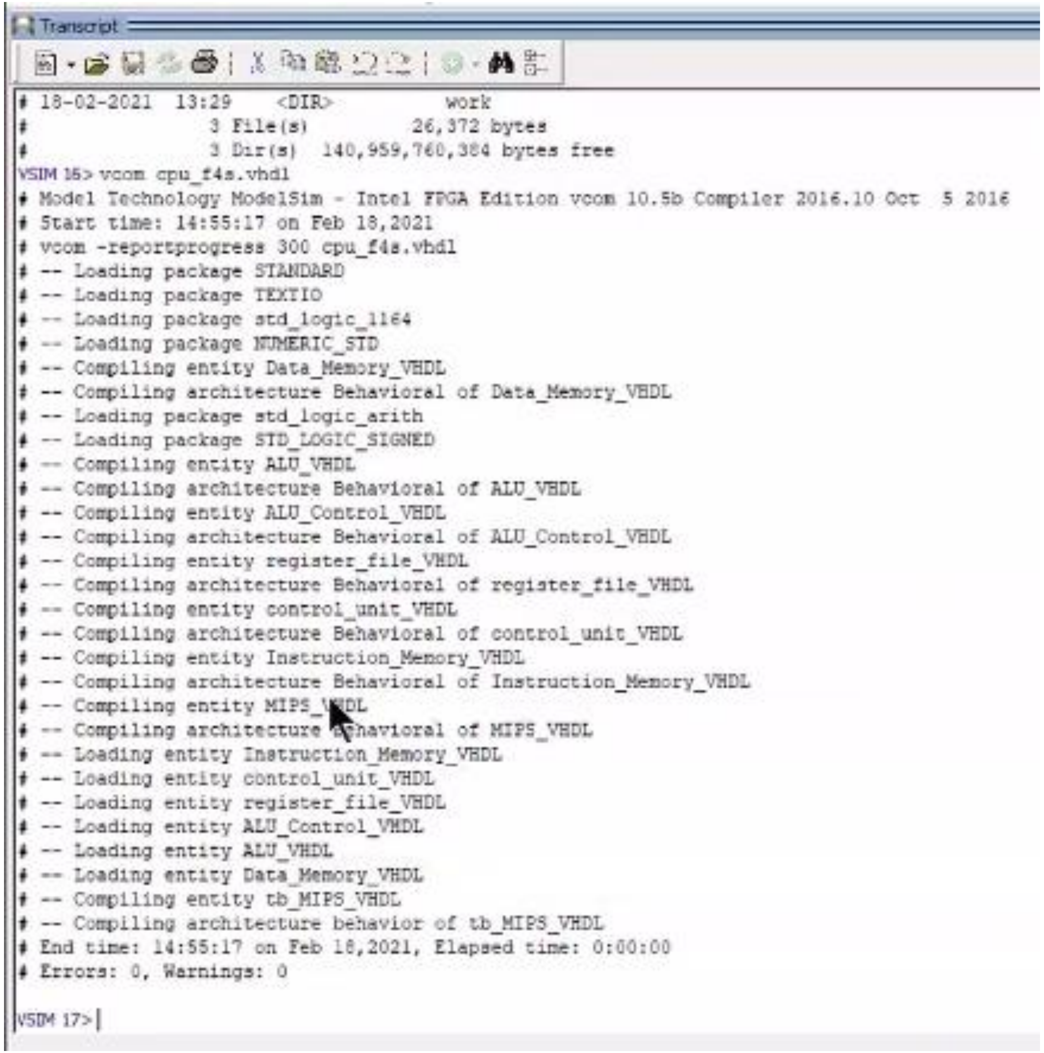
Sign Extended version of imm field


RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instr[15:13]
Instr[12:10]
Instr[9:7]
Instr[6:4]
7
Read data 1
Read data 2
Write register
Write data
Registers
Instr[6:0]
Sign extend
ALU
Zero
ALU control block
ALUsrc
Mux

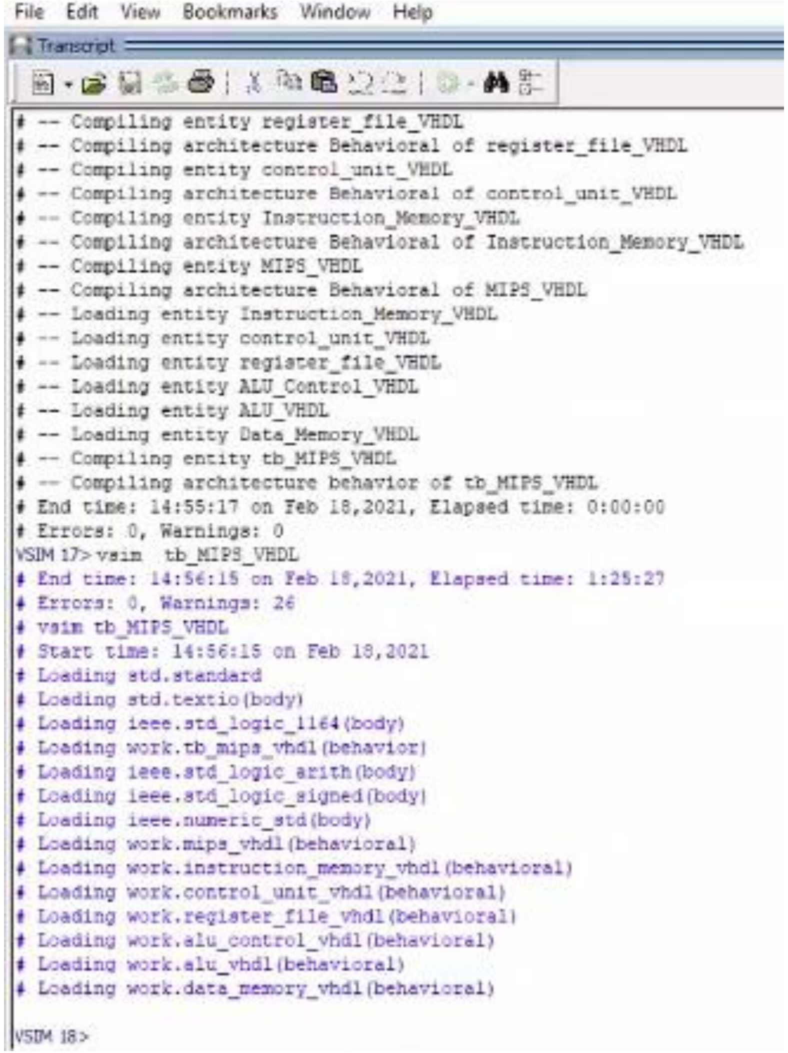
t.com



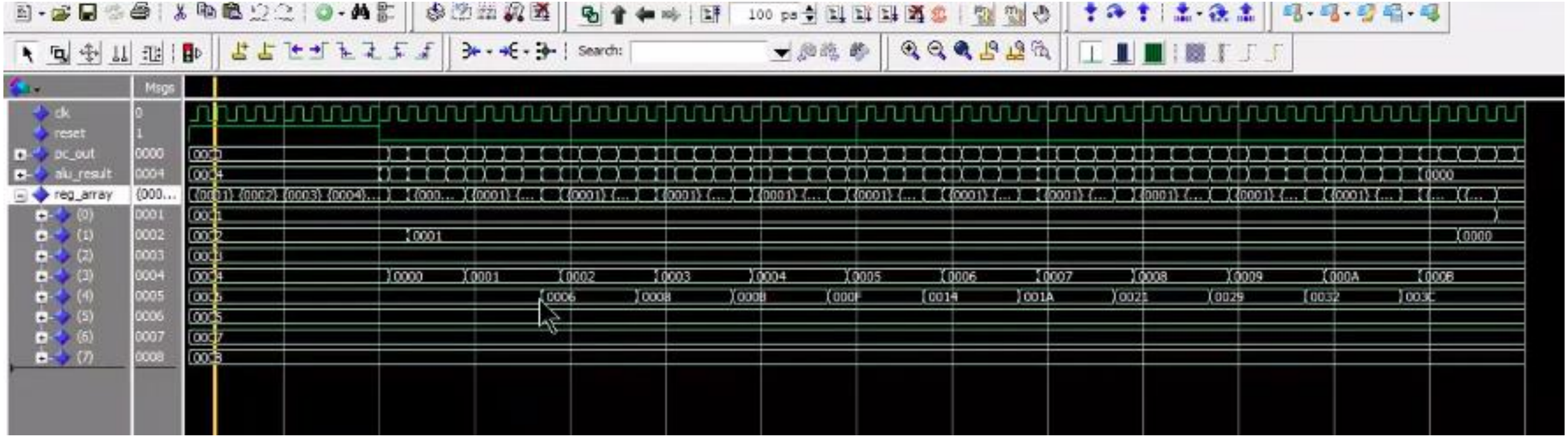
```
-- sign extend
tmp1 <= (others => instr(6));
sign_ext_im <=  tmp1 & instr(6 downto 0);
zero_ext_im <= "000000000"& instr(6 downto 0);
imm_ext <= sign_ext_im when sign_or_zero='1' else zero_ext_im;
```



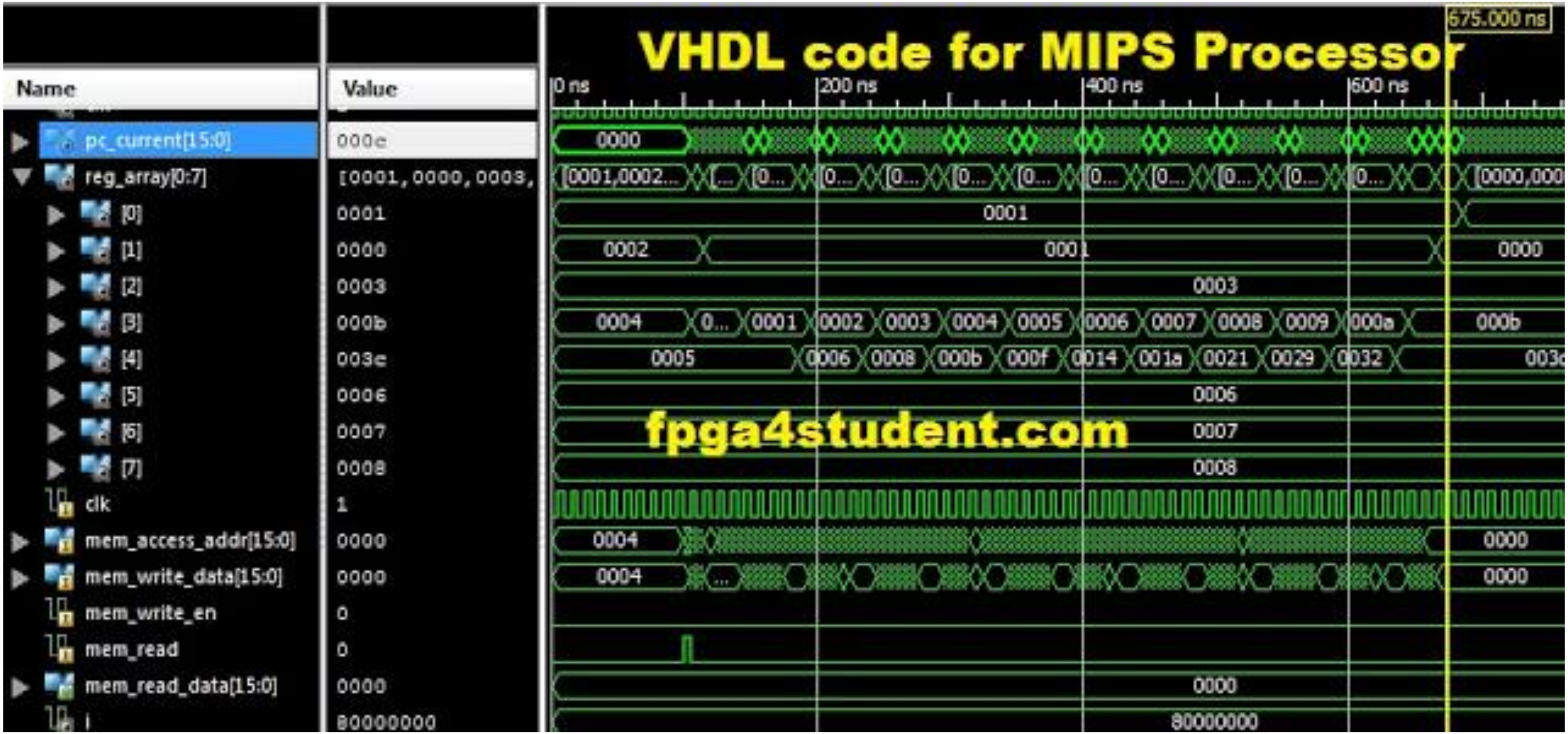
```
# 18-02-2021  13:29    <DIR>          work
#                3 File(s)         26,372 bytes
#                3 Dir(s)  140,959,760,384 bytes free
VSIM 16> vcom cpu_f4s.vhdl
# Model Technology ModelSim - Intel FPGA Edition vcom 10.5b Compiler 2016.10 Oct  5 2016
# Start time: 14:55:17 on Feb 18,2021
# 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>
```



```
# -- 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>
```



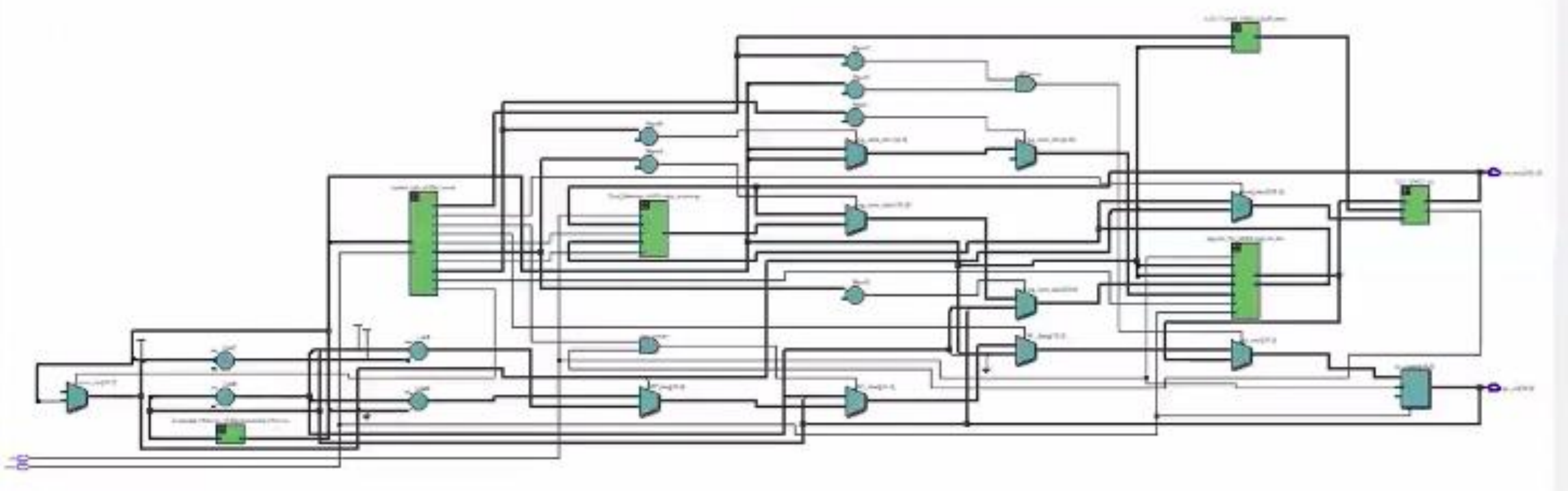
```
clk          0
reset        1
pc_out       0000
alu_result   0004
reg_array    {000...
  (0)        0001
  (1)        0002
  (2)        0003
  (3)        0004
  (4)        0005
  (5)        0006
  (6)        0007
  (7)        0008
```



VHDL code for MIPS Processor

fpga4student.com

Name	Value
pc_current[15:0]	000e
reg_array[0:7]	[0001,0000,0003,
[0]	0001
[1]	0000
[2]	0003
[3]	000b
[4]	003e
[5]	0006
[6]	0007
[7]	0008
clk	1
mem_access_addr[15:0]	0000
mem_write_data[15:0]	0000
mem_write_en	0
mem_read	0
mem_read_data[15:0]	0000
i	80000000



To Be Continued (and revisited)