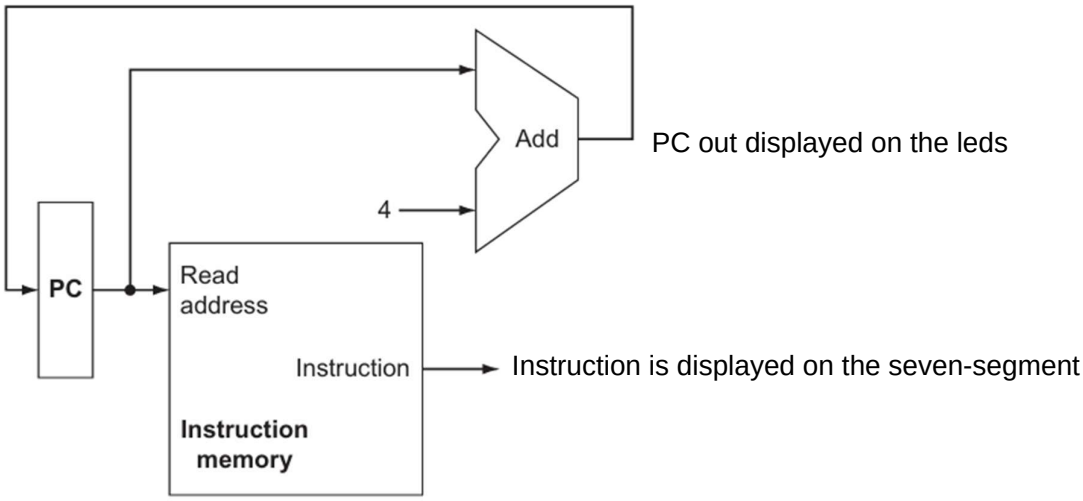
Fetch Module



***Design 1**: A portion of the datapath used for fetching instructions and incrementing the program counter.*

ENTITIES:

1:Seven Segment Display

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY sevenSegement IS
    PORT (
        bininput : IN std_logic_vector(3 DOWNTO 0);
        cathodes : OUT std_logic_vector(6 DOWNTO 0)
    );
END sevenSegement;

architecture Behavioural of sevenSegement is
begin
    cathodes <= "1000000" when (bininput = "0000") else
                "1111001" when (bininput = "0001") else
                "0100100" when (bininput = "0010") else
                "0110000" when (bininput = "0011") else
                "0011001" when (bininput = "0100") else
                "0010010" when (bininput = "0101") else
                "0000010" when (bininput = "0110") else
                "1111000" when (bininput = "0111") else
                "0000000" when (bininput = "1000") else
                "0010000" when (bininput = "1001") else
                "0001000" when (bininput = "1010") else
                "0000011" when (bininput = "1011") else
                "1000110" when (bininput = "1100") else
                "0100001" when (bininput = "1101") else
                "0000110" when (bininput = "1110") else
                "0001110" when (bininput = "1111") else
                "1111111";
end Behavioural;
```

2:Fetch

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```vhdl
  use IEEE.NUMERIC_STD.ALL;
  use ieee.std_logic_unsigned.all;
use work.MyPackage.all;

entity fetch is
    port (
        PC_out                      : out STD_LOGIC_VECTOR (31 downto 0);
        instruction                 : out STD_LOGIC_VECTOR (31 downto 0);
        branch_addr,jump_addr     : in STD_LOGIC_VECTOR (31 downto 0);
        branch_decision,jump_decision,reset,clock  : in std_logic
    );
end fetch;

architecture bhv of fetch is

    -- Define instruction memory array type
    type mem_array is array(0 to 15) of std_logic_vector(31 downto 0);

begin

    -- Process for handling the fetch operation
    process(clock, reset)

        -- Define variables inside the process
        variable pc     : std_logic_vector(31 downto 0) ;
        variable index  : integer range 0 to 15 := 0;

        -- Initialize the instruction memory with machine code
        variable mem    : mem_array := (
            X"8c220000", -- lw $2, 0($1)     -- Load word       -- 1
            X"8c640001", -- lw $4, 1($3)     -- Load word       -- 2
            X"00822022", -- sub $4, $4, $3   -- Subtract        -- 3
            X"ac400000", -- sw $4, 0($3)     -- Store word      -- 4
            X"1022fffa", -- beq $1, $2, L    -- Branch if equal -- 5
            X"00612024", -- and $4, $3, $1   -- AND operation   -- 6
            X"08000000", -- j L              -- Jump            -- 7
            X"00000000", -- NOTHING          -- No operation    -- 8
            others => X"00000000"  -- Initialize remaining locations
        );

    begin
        -- fetch Process--

        -- Check if reset signal is active
            if reset = '1' then
            pc := (others => '0');
            instruction <= (others => '0');
            PC_out <= (others => '0');
            elsif rising_edge(clock) then
            -- Handle branch/jump
            if branch_decision = '1' then
                pc := branch_addr;
            elsif jump_decision = '1' then
                pc := jump_addr;
                end if;
            end if;

            -- Calculate memory index from PC (using word alignment)
            index := to_integer(unsigned(pc(3 downto 0)));
            -- Increment PC by 1(4-bit) (after fetching)
            pc := pc + "1";
```

```
            -- Fetch instruction from memory
            instruction <= mem(index);
            -- Featch program counter wrt
            PC_out <= pc;

    end process;
end bhv;
```

PACKAGE:

```
library ieee;
use ieee.std_logic_1164.all;

package MyPackage is

    component sevenSegement is
        port (
            bininput : in  std_logic_vector(3 downto 0);
            cathodes : out std_logic_vector(6 downto 0)
        );
    end component;

    component fetch is
        port (
            PC_out          : out STD_LOGIC_VECTOR (31 downto 0);
            instruction        : out STD_LOGIC_VECTOR (31 downto 0);
            branch_addr,
  jump_addr         : in STD_LOGIC_VECTOR (31 downto 0);
            branch_decision,
  jump_decision,
     reset,clock    : in std_logic
        );
    end component;

end package MyPackage;
```

TOP-ENTITY(Wrapper File)

```
-- Library declarations
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;  -- Single library for arithmetic and conversion
use work.MyPackage.all;

-- Entity Declaration
entity fetchModule is
    port(
        hex0,hex1,hex2,hex3,
  hex4,hex5,hex6,hex7     : out std_logic_vector(6 downto 0);
        leds                    : out std_logic_vector(3 downto 0);
        topclock,topreset       :in std_logic
    );
end fetchModule;


architecture bhv of fetchModule is
    -- TOP INSTRUCTION <- show instruction present in register file
    -- TOP PC <- show address counter of above information

    signal top_instruction,top_pcout : std_LOGIC_VECTOR(31 downto 0);

begin
```

```
    -- Progaram Counter
    f1: fetch port map (
        PC_out              => top_pcout,
        instruction         => top_instruction,
        branch_addr         => X"00000000",
        jump_addr           => X"00000000",
        branch_decision     => '0',  -- branch_des
        jump_decision       => '0',  -- jump_des
        reset               => topreset,
        clock               => topclock
    );
     leds <= top_pcout(3 downto 0);


     -- Instruction
    u7: sevenSegement port map (
        bininput => top_instruction(31 downto 28),
        cathodes => hex7
    );
    u6: sevenSegement port map (
        bininput => top_instruction(27 downto 24),
        cathodes => hex6
    );
    u5: sevenSegement port map (
        bininput => top_instruction(23 downto 20),
        cathodes => hex5
    );
    u4: sevenSegement port map (
        bininput => top_instruction(19 downto 16),
        cathodes => hex4
    );
    u3: sevenSegement port map (
        bininput => top_instruction(15 downto 12),
        cathodes => hex3
    );
    u2: sevenSegement port map (
        bininput => top_instruction(11 downto 8),
        cathodes => hex2
    );
    u1: sevenSegement port map (
        bininput => top_instruction(7 downto 4),
        cathodes => hex1
    );
    u0: sevenSegement port map (
        bininput => top_instruction(3 downto 0),
        cathodes => hex0
    );

end bhv;
```

On click actions :

SELECTION	OPERATION
Sliding_switch_01	Reset
Push_button_01	Next Information

Output :

SELECTION	OPERATION
Seven_segment	Instruction
Leds	Pc_out