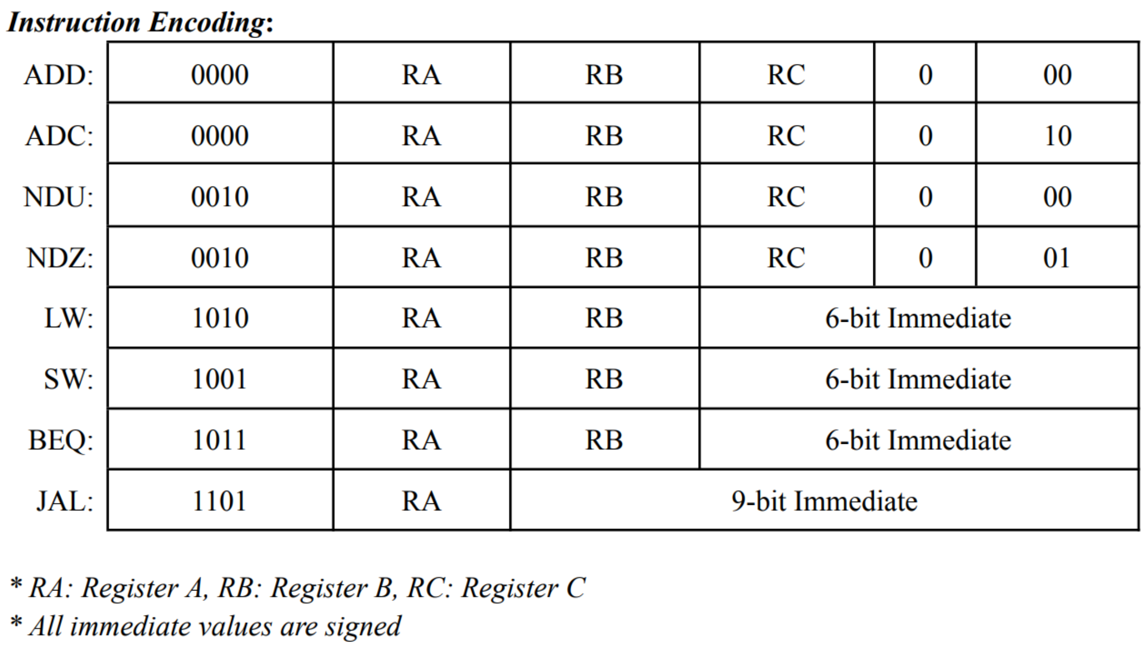
Assignment-1

Shelly Anissa
B210434CS

INTRODUCTION

NITC-RISC24 is a 16 bit processor, which is capable of executing instructions of format R, I and J. The encoding of each instruction is shown below.



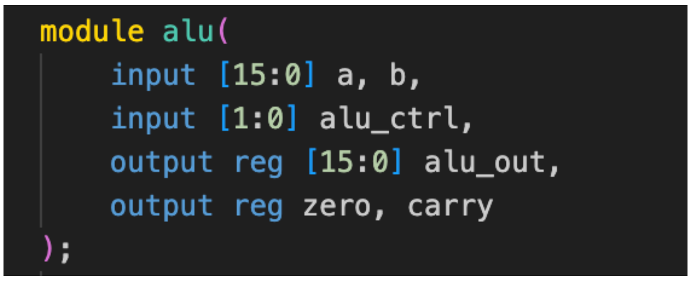
***Instruction Encoding*:**

ADD:	0000	RA	RB	RC	0	00
ADC:	0000	RA	RB	RC	0	10
NDU:	0010	RA	RB	RC	0	00
NDZ:	0010	RA	RB	RC	0	01
LW:	1010	RA	RB	6-bit Immediate		
SW:	1001	RA	RB	6-bit Immediate		
BEQ:	1011	RA	RB	6-bit Immediate		
JAL:	1101	RA	9-bit Immediate			

* *RA: Register A, RB: Register B, RC: Register C*
* *All immediate values are signed*

MODULES

1. ALU

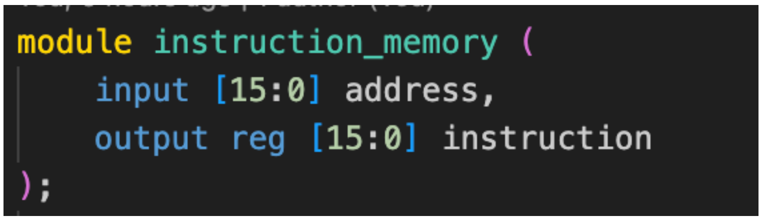


```
module alu(
    input [15:0] a, b,
    input [1:0] alu_ctrl,
    output reg [15:0] alu_out,
    output reg zero, carry
);
```

The ALU takes in two operands. The operation to be performed is dictated by the alu_ctrl signal. To accommodate all the 8 instructions the ALU need only perform three operations ADD, SUBTRACT and NAND.

ALU_CTRL	ALU OPERATION	INSTRUCTION
00	ADD	ADD, ADC, LW, SW
01	NAND	NDU, NDZ
10	SUB	BEQ

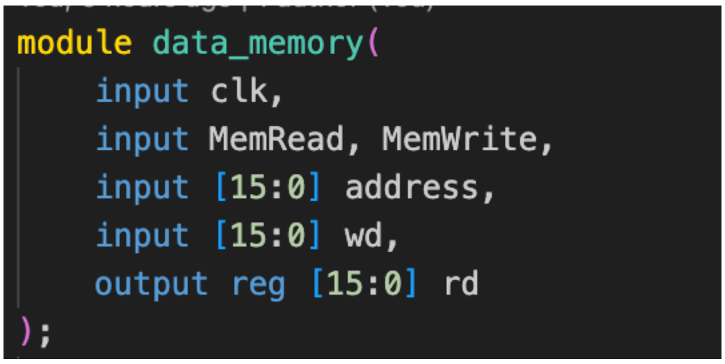
2. INSTRUCTION MEMORY



```verilog
module instruction_memory (
    input [15:0] address,
    output reg [15:0] instruction
);
```

This module is responsible for reading the instructions from file onto instruction memory upon initialisation. The output of the module would be the instruction present at the specified address.

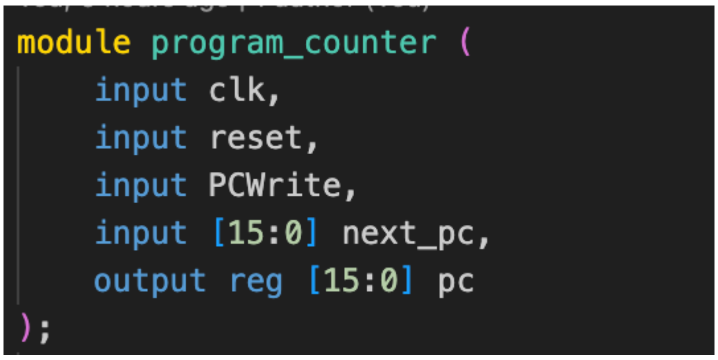
3. DATA MEMORY



```verilog
module data_memory(
    input clk,
    input MemRead, MemWrite,
    input [15:0] address,
    input [15:0] wd,
    output reg [15:0] rd
);
```

The module data memory is responsible for writing or reading from memory based on the input signals clk, MemWrite and MemRead. The data to be written is present in the wd register and data read from memory would be present in rd register.

4. PROGRAM COUNTER



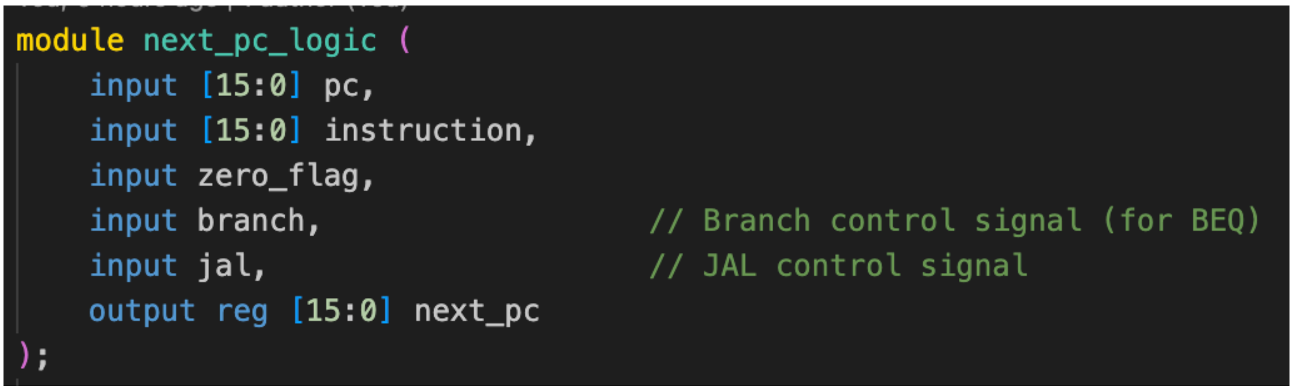
```verilog
module program_counter (
    input clk,
    input reset,
    input PCWrite,
    input [15:0] next_pc,
    output reg [15:0] pc
);
```

PCWrite signal enables the update of pc with the value next_pc. If the reset signal is set to high, program counter would be assigned the value 0 and thus resetting the program execution.

5. NEXT PC LOGIC

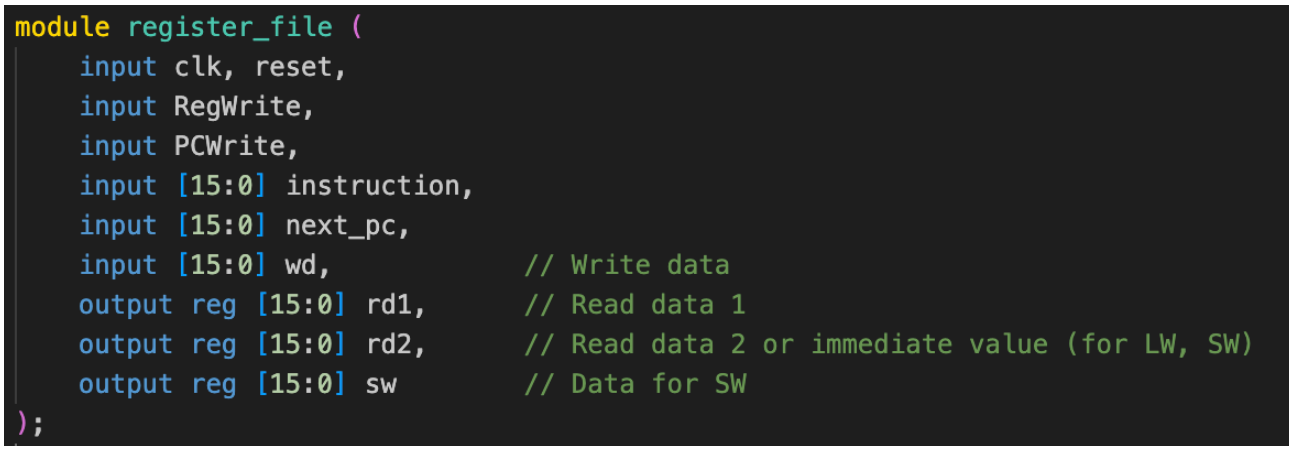
The next pc logic is computed differently for different values of the Branch, Zero and JAL input signals.

i) JAL: next_pc = pc + sign_extended offset (6 bits)
ii) BRANCH (zero flag is also set) : next_pc = pc + sign_extended offset (9 bits)
iii) DEFAULT: next_pc = pc + 1



```
module next_pc_logic (
    input [15:0] pc,
    input [15:0] instruction,
    input zero_flag,
    input branch,              // Branch control signal (for BEQ)
    input jal,                 // JAL control signal
    output reg [15:0] next_pc
);
```

6. REGISTER FILE

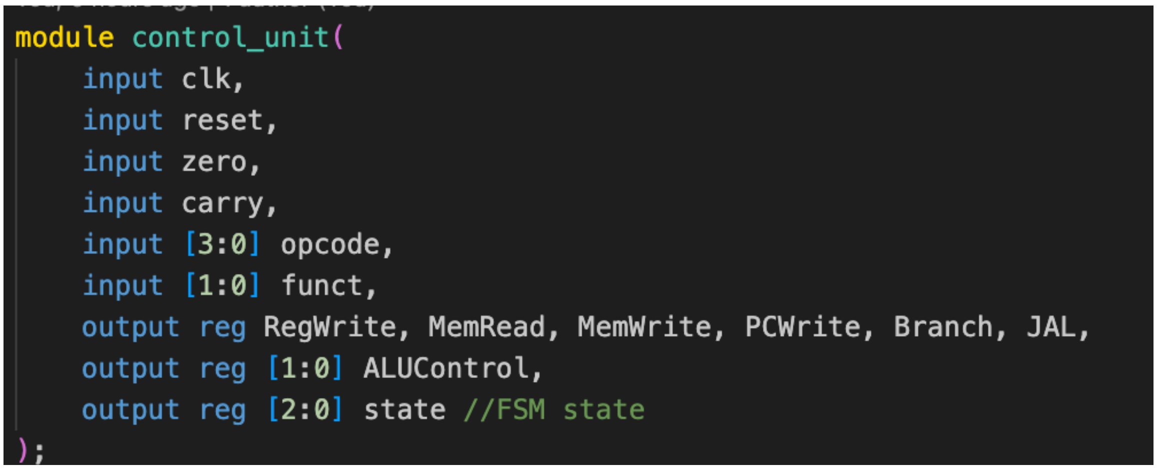


```
module register_file (
    input clk, reset,
    input RegWrite,
    input PCWrite,
    input [15:0] instruction,
    input [15:0] next_pc,
    input [15:0] wd,           // Write data
    output reg [15:0] rd1,     // Read data 1
    output reg [15:0] rd2,     // Read data 2 or immediate value (for LW, SW)
    output reg [15:0] sw       // Data for SW
);
```

The source and destination registers are determined from the instruction based on the opcode. Registers rd1 and rd2 have the contents of source registers in R type instruction and register content and immediate value in the case of I type instruction. Register sw contains the word to be written to memory for store word instruction.

RegWrite enables writing onto register, excluding R0.
PCWrite signal is used to update R0 with the program counter value.

7. CONTROL UNIT



```
module control_unit(
    input clk,
    input reset,
    input zero,
    input carry,
    input [3:0] opcode,
    input [1:0] funct,
    output reg RegWrite, MemRead, MemWrite, PCWrite, Branch, JAL,
    output reg [1:0] ALUControl,
    output reg [2:0] state //FSM state
);
```

The control unit is responsible for generating the control signals namely RegWrite, MemRead, MemWrite, PCWrite, JAL, Branch, ALUControl.

The following encoding is used to denote the phase of the multi cycle execution:
IF = 3'b000, ID = 3'b001, EX = 3'b010, MEM = 3'b011, WB = 3’b100;

Given below is the state transition logic.

```
always @(posedge clk or posedge reset) begin
    if (reset)
        state <= IF;
    else begin
        case (state)
            IF: state <= ID;
            ID: if (opcode == 4'b1101) state <= WB; // JAL
                else state <= EX;
            EX: if (opcode == 4'b1010 || opcode == 4'b1001) state <= MEM; //LW, SW
                else state <= WB;
            MEM: state <= WB;
            WB: state <= IF;
        endcase
    end
end
```

The control signals for different phases of an I-Type instruction **LW R1 R2 Imm**

PHASE	MEMWRITE	MEMREAD	REGWRITE	PCWRITE	JAL	BRANCH	ALU_CTRL
IF	0	0	0	1	0	0	xx
ID	0	0	0	0	0	0	xx
EX	0	0	0	0	0	0	00
MEM	0	1	0	0	0	0	00
WB	0	0	1	0	0	0	00

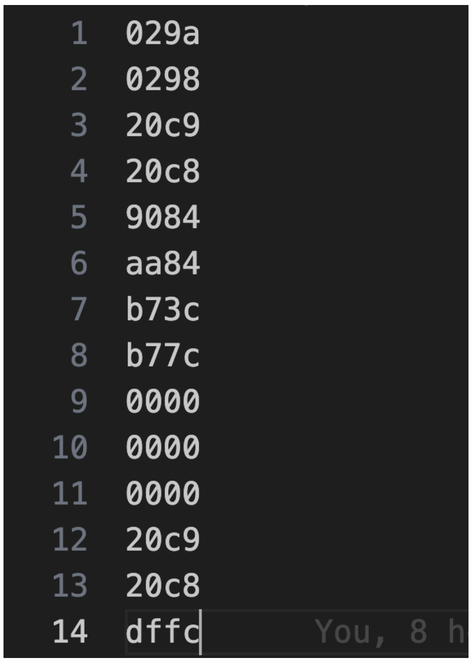
8. NITC_RISC24_PROCESSOR

```
module nitc_risc24_processor(
    input clk,
    input reset
);
```

This top level module is responsible for instantiating all the submodules, declaring wires and registers. It multiplexes the write data for register file as follows:
i) LW: rw of data_memory
ii) JAL: pc + 1
iii) DEFAULT: alu_out

INSTRUCTION RUN

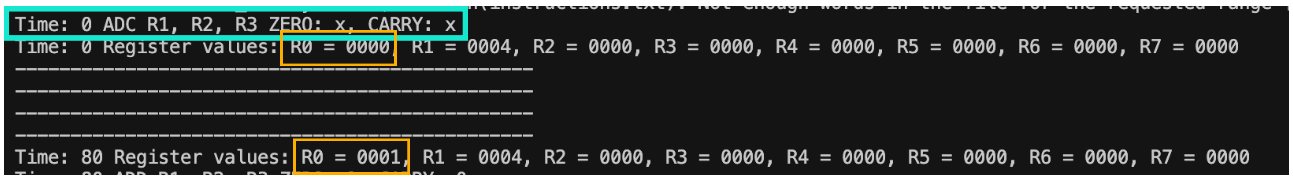
Given below is the sample run of the processor with the contents of instruction file being as shown.



```
029a
0298
20c9
20c8
9084
aa84
b73c
b77c
0000
0000
0000
20c9
20c8
dffc
```

The state of the registers at the time of fetch of each instruction is displayed alongside the time instant. The instruction under execution is highlighted within cyan block. The line separations '----' indicate the number of cycles/stages elapsed within the execution of the instruction. The clock signal toggles every 10 ns. That is a clock cycle is 20 ns long. Register R1 is initialised with the value 4, whereas all the other registers are 0.

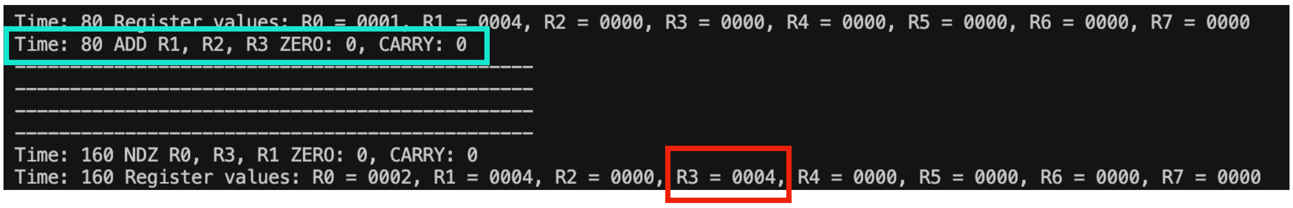
1. ADC



```
Time: 0 ADC R1, R2, R3 ZERO: x, CARRY: x
Time: 0 Register values: R0 = 0000, R1 = 0004, R2 = 0000, R3 = 0000, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
------------------------------------------------
------------------------------------------------
------------------------------------------------
------------------------------------------------
Time: 80 Register values: R0 = 0001, R1 = 0004, R2 = 0000, R3 = 0000, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
```

alu_out = R1 + R2 => 4 + 0 => 4
However the value is not written back to R3 since carry flag is not set.
The program counter R0 is incremented.

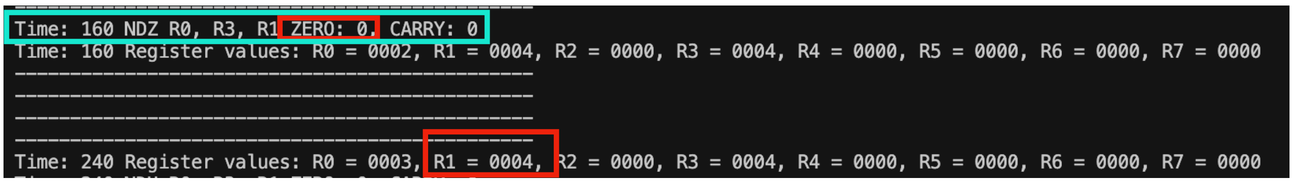
2. ADD



```
Time: 80 Register values: R0 = 0001, R1 = 0004, R2 = 0000, R3 = 0000, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
Time: 80 ADD R1, R2, R3 ZERO: 0, CARRY: 0
------------------------------------------------
------------------------------------------------
------------------------------------------------
------------------------------------------------
Time: 160 NDZ R0, R3, R1 ZERO: 0, CARRY: 0
Time: 160 Register values: R0 = 0002, R1 = 0004, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
```

alu_out = R1 + R2 => 4
The value 4 is written back to R3.
It could also be noticed that the program counter value at R0 is also updated.

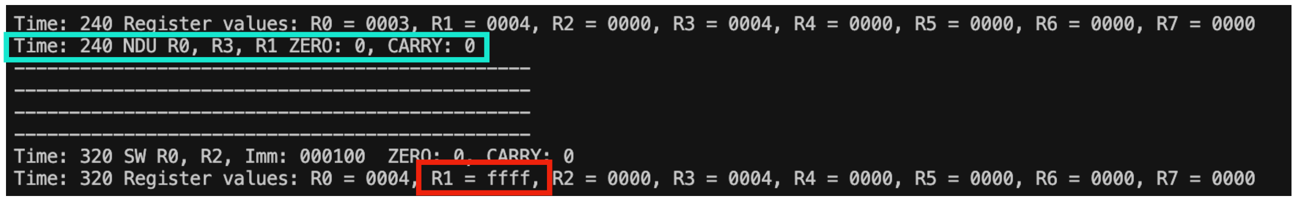
3. NDZ



```
Time: 160 NDZ R0, R3, R1 ZERO: 0, CARRY: 0
Time: 160 Register values: R0 = 0002, R1 = 0004, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 240 Register values: R0 = 0003, R1 = 0004, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
```

alu_out = ~(R0 & R3) => ~(0000 & 0003) => ffff
However write back to R1 does not happen since Zero flag is not set
Program counter updated to 3

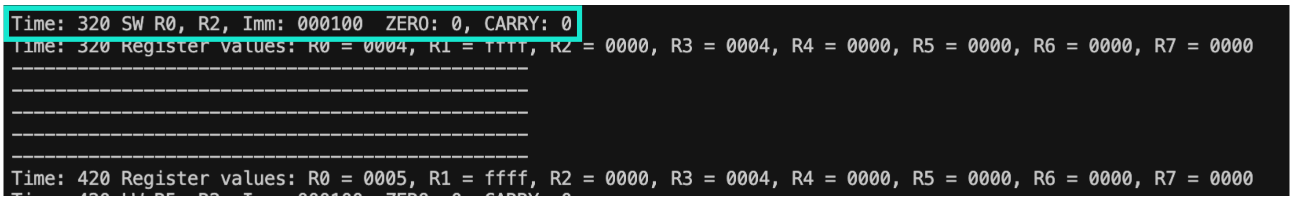
4. NDU



```
Time: 240 Register values: R0 = 0003, R1 = 0004, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
Time: 240 NDU R0, R3, R1 ZERO: 0, CARRY: 0
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 320 SW R0, R2, Imm: 000100  ZERO: 0, CARRY: 0
Time: 320 Register values: R0 = 0004, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
```

alu_out = ~(R0 & R3) => ~(0000 & 0003) => ffff
ffff is written back to R1
Program counter updated to 4

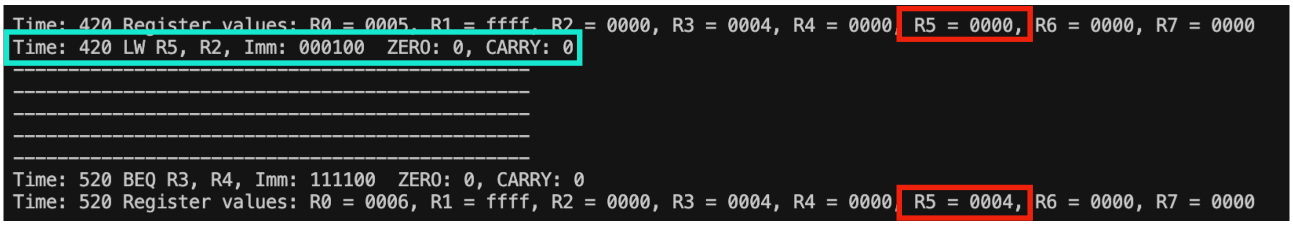
5. SW



```
Time: 320 SW R0, R2, Imm: 000100  ZERO: 0, CARRY: 0
Time: 320 Register values: R0 = 0004, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 420 Register values: R0 = 0005, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
```

alu_out = R2 + sign_extend(000100) => 4
Contents of R0 ie 0004 is written at data address 16'b 4
It has to be also noted that the number of cycles/stages elapsed here is 5 unlike 4 in R type instructions, due to the MEM stage present.
Program counter updated to 5

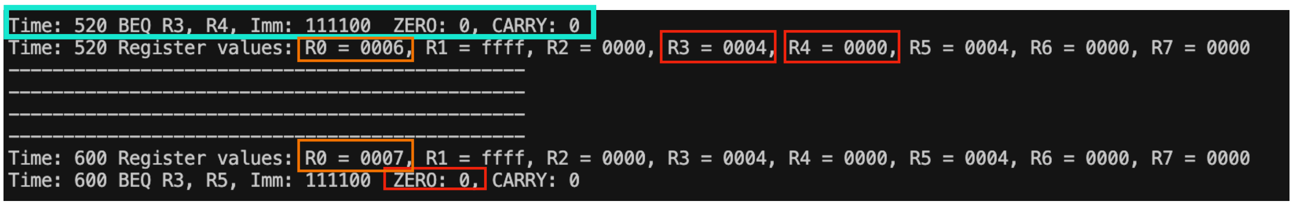
6. LW



```
Time: 420 Register values: R0 = 0005, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0000, R6 = 0000, R7 = 0000
Time: 420 LW R5, R2, Imm: 000100  ZERO: 0, CARRY: 0
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 520 BEQ R3, R4, Imm: 111100  ZERO: 0, CARRY: 0
Time: 520 Register values: R0 = 0006, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 0000
```

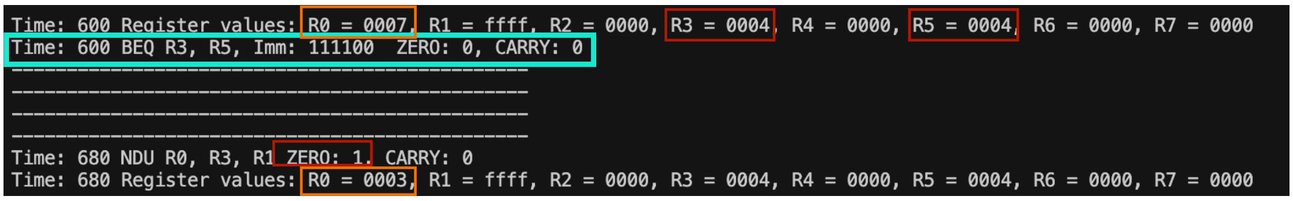
Data stored at address 4 in previous instruction is fetched and written onto R5
alu_out = R2 + sign_extend(00100) => 4
5 written onto R5
Program counter incremented to 6
The instruction goes through 5 stages indicated by the number of dashed lines.

7. BEQ (not taken)


```
Time: 520 BEQ R3, R4, Imm: 111100  ZERO: 0, CARRY: 0
Time: 520 Register values: R0 = 0006, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 0000
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 600 Register values: R0 = 0007, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 0000
Time: 600 BEQ R3, R5, Imm: 111100  ZERO: 0, CARRY: 0
```

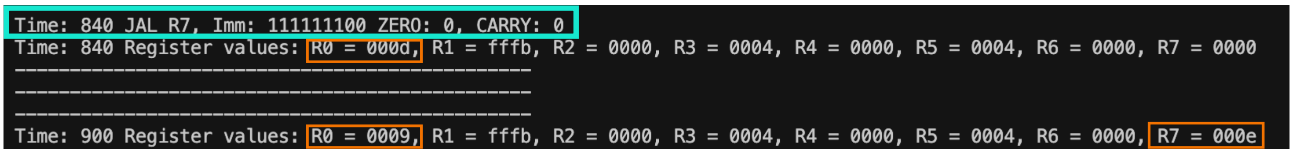
alu_out = R3 - R4 => 0004 - 0000 => 0004
Thus zero flag is not set and branch is not taken.
Program counter incremented as always.

8. BEQ (taken)


```
Time: 600 Register values: R0 = 0007, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 0000
Time: 600 BEQ R3, R5, Imm: 111100  ZERO: 0, CARRY: 0
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 680 NDU R0, R3, R1 ZERO: 1, CARRY: 0
Time: 680 Register values: R0 = 0003, R1 = ffff, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 0000
```

alu_out = R3 - R5 => 0004 - 0004 => 0000
Thus zero flag is set and branch is taken.
Branch address = PC + sign_extend(offset) => 0007 + fffc(two's comp of -4) => 3
PC is set to the branch target 3

9. JAL


```
Time: 840 JAL R7, Imm: 111111100 ZERO: 0, CARRY: 0
Time: 840 Register values: R0 = 000d, R1 = fffb, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 0000
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
Time: 900 Register values: R0 = 0009, R1 = fffb, R2 = 0000, R3 = 0004, R4 = 0000, R5 = 0004, R6 = 0000, R7 = 000e
```

Jump address = PC + sign_extend(offset) => 000d + fffc(two's comp of -4) => 9
Link address = PC + 1 => d + 1 => e
Link address is written back to R7