FPGA Implementation of 32-bit FSM-based Multi-State MIPS Processor

ES215 - Computer Organization and Architecture

TEAM - 22 ALUminati

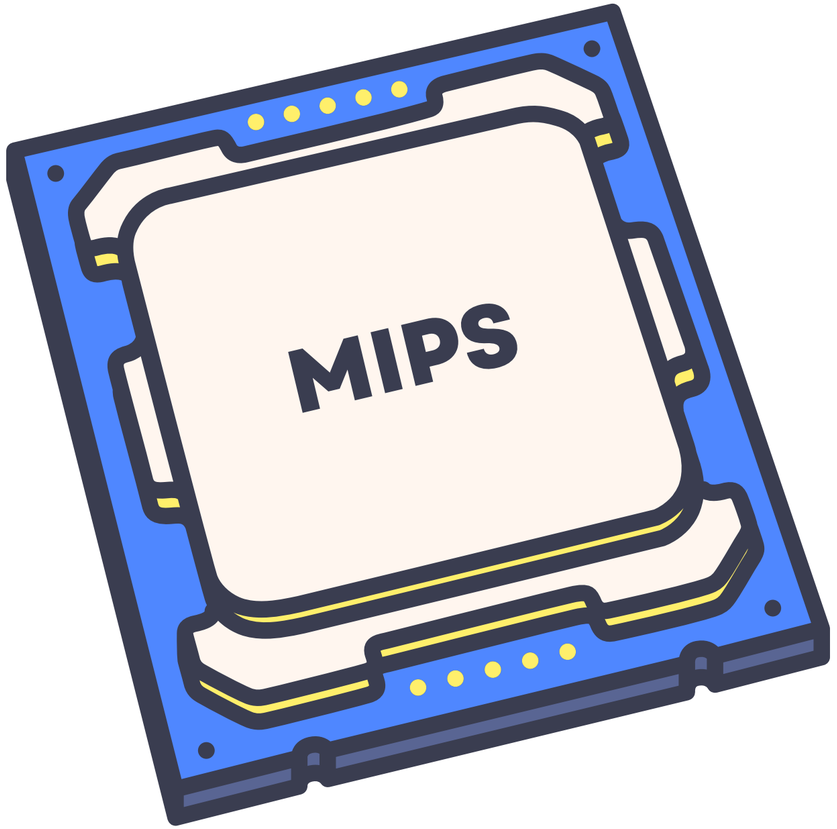
Guntas Singh Saran (22110089)

Hrriday V. Ruparel (22110099)

Kishan Ved (22110122)

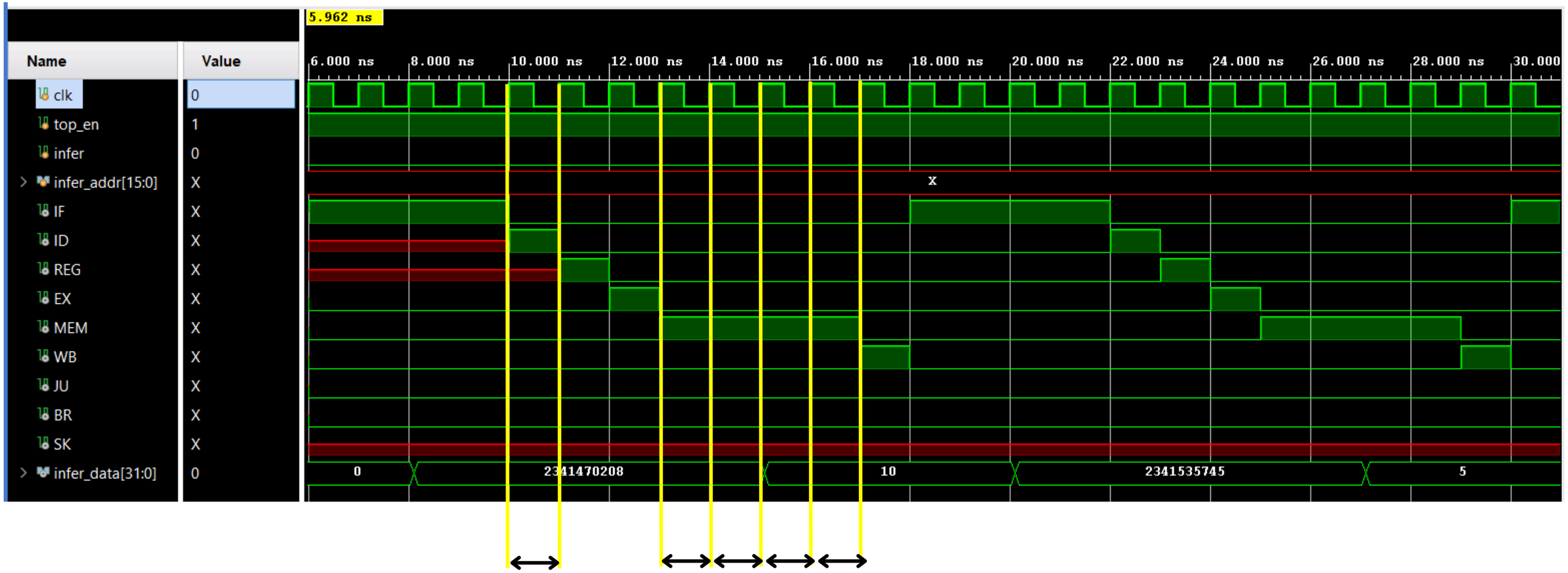
Pranav Patil (22110199)

Indian Institue of Technology Gandhinagar
Palaj, Gujarat - 382355



Source: https://github.com/guntas-13/mips-processor-basys3

Multi-Cycle? Multi-State?



Here Stage (ID) itself is a State

1 Stage (IF) composed of 4 states

Multi-Cycle? Multi-State?

Memory => Stage

Made up of 4 *States*

Clock Cycle Tc is then the time of *longest state* made of combinational circuit

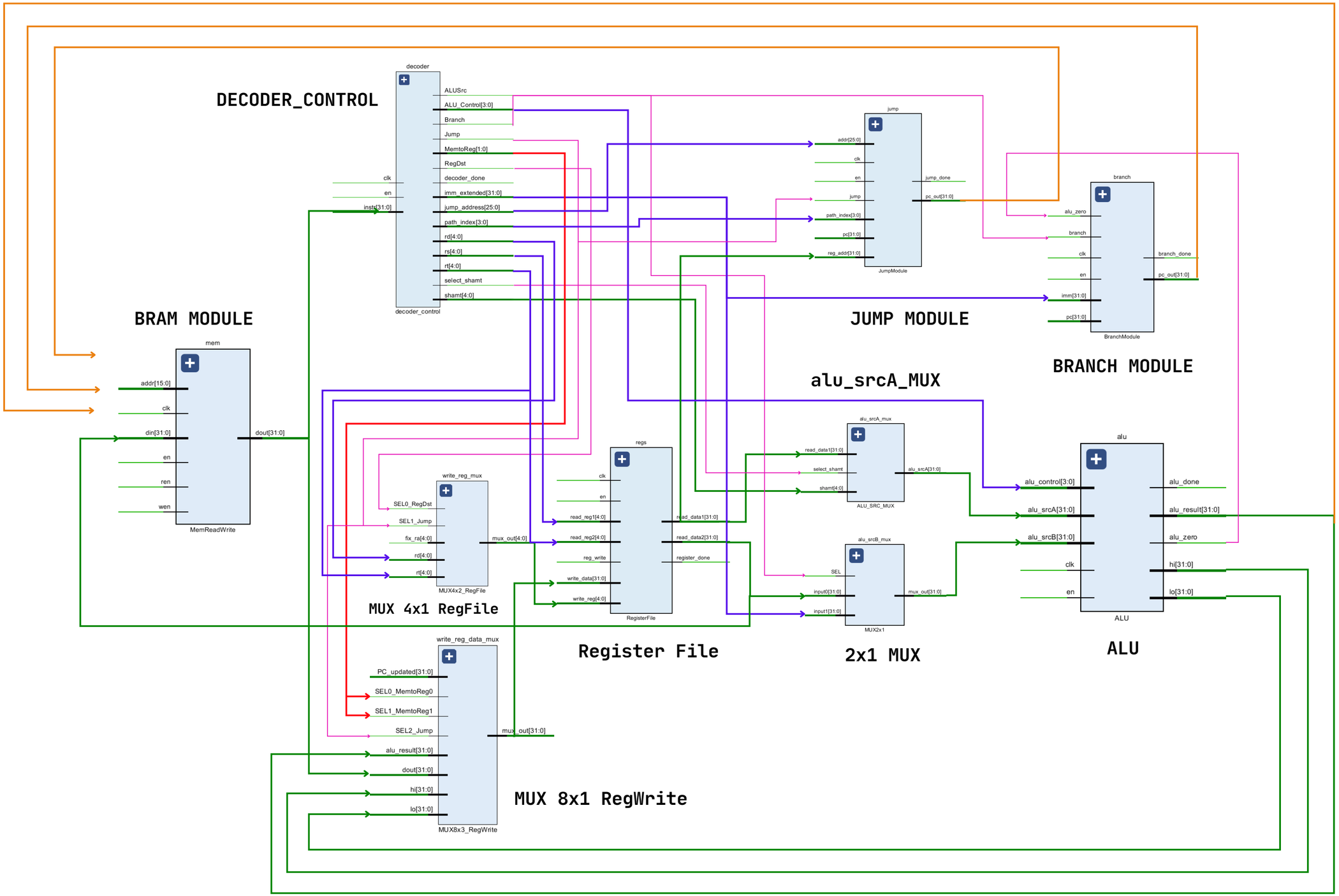
```
case(state)
    IDLESRC: begin
        if (top_en) state <= FETCH;
        else state <= IDLESRC;
    end
    FETCH: begin
        state <= RED1;
        mem_en <= 1;
        mem_ren <= 1;
        mem_wen <= 0;
        mem_addr <= pc[15:0];
        IF <= 1;
        EX <= 0;
        MEM <= 0;
        WB <= 0;
        JU <= 0;
        BR <= 0;
        SK <= 0;
    end
    RED1: begin
        state <= RED2;
    end
    RED2: begin
        state <= RED3;
    end
    RED3: begin
        state <= DECODE;
        instr_reg <= mem_dout;
    end
```

Processor Specification

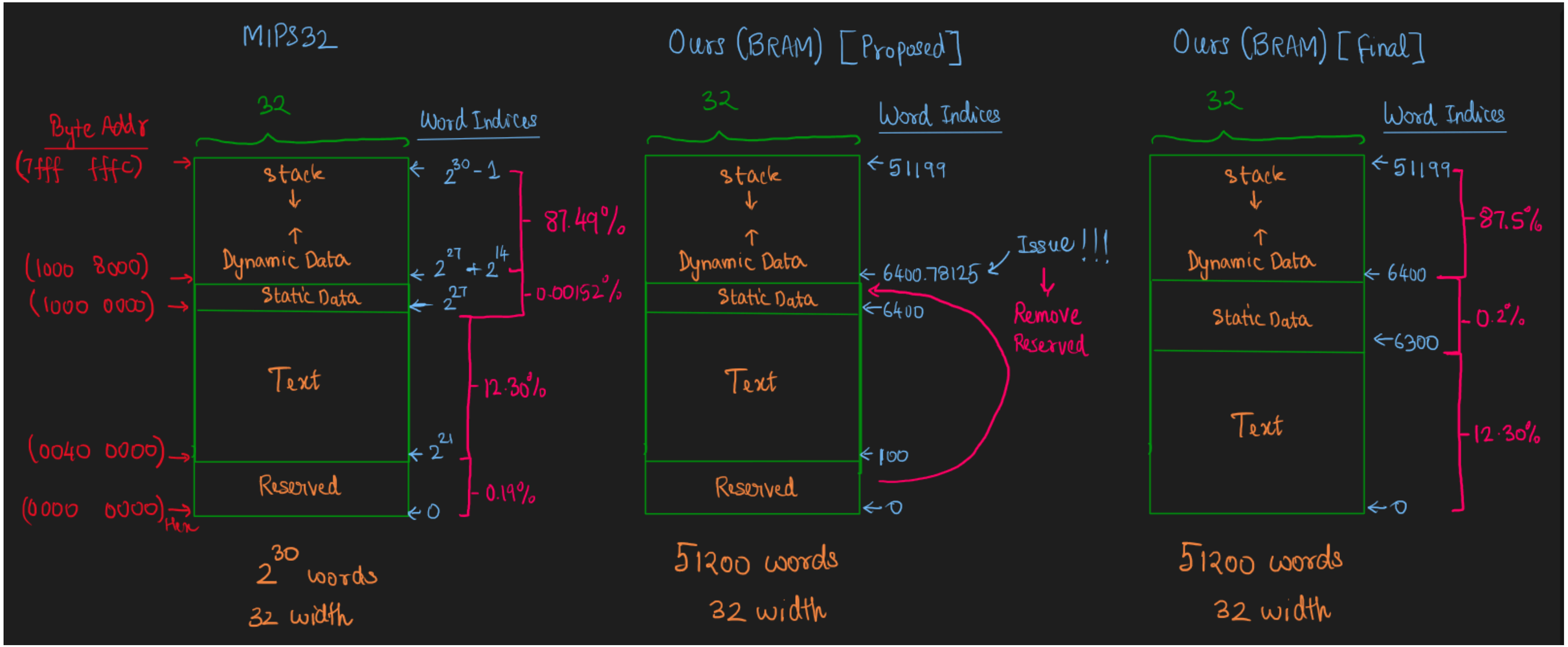
- Clock-edge and Control signal triggered states
- **FSM-based** processor implementation with each stage composed of one or more states
- Every state has a **combinatorial circuit** made of modules
- **Integer Register** Operations
- **Word-Addressing** (So no byte-level index)
- 2’s complement System
- Register File on LUTs while Memory on BRAM
- Modules as mentioned in the slides ahead
- All **32 Registers** as specified by MIPS

Our ISA

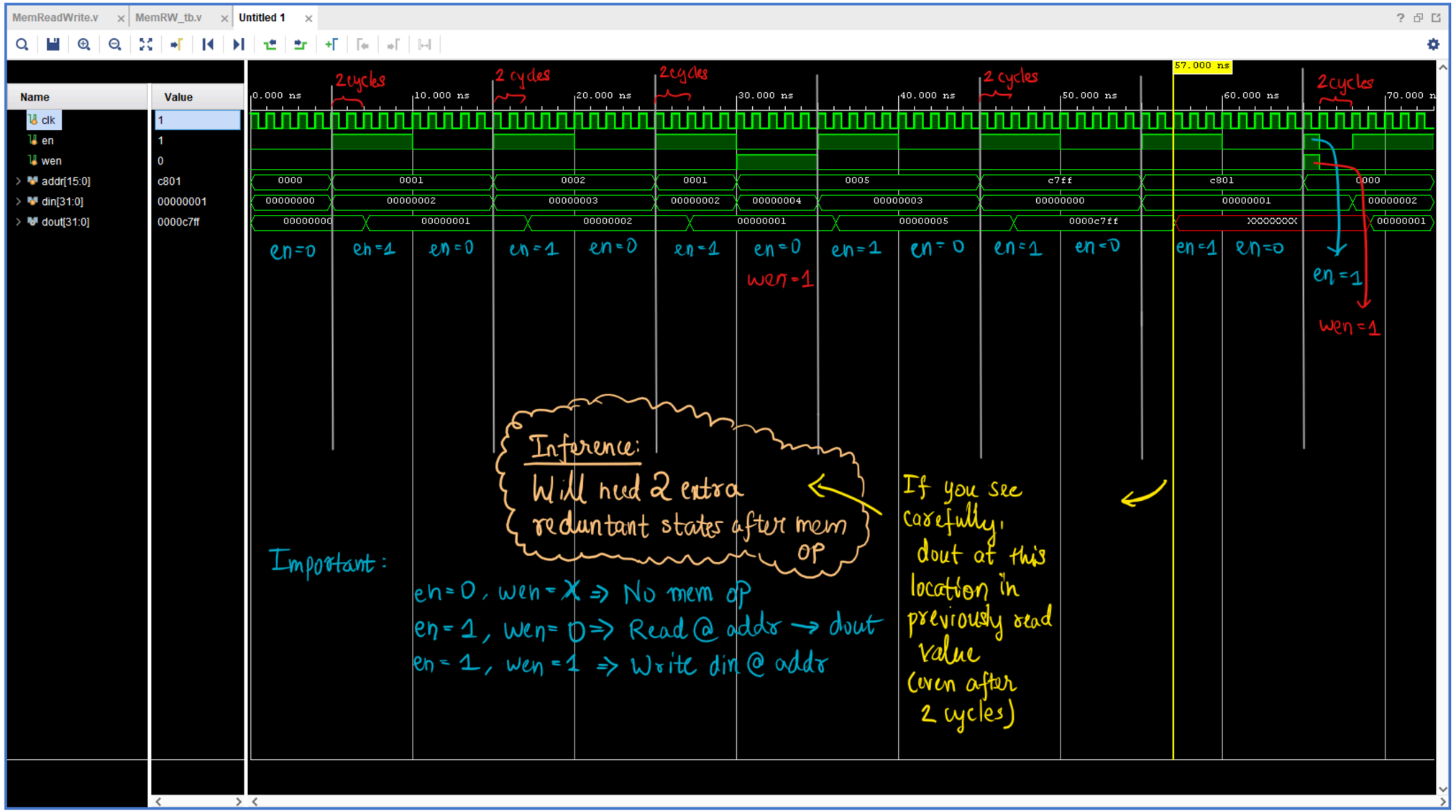
Instruction	Format	Operation
sll $rd, $rt, shamt	R-Type	R[$rd] ← R[$rt] << shamt
srl $rd, $rt, shamt	R-Type	R[$rd] ← R[$rt] >> shamt
add $rd, $rs, $rt	R-Type	R[$rd] ← R[$rs] + R[$rt]
sub $rd, $rs, $rt	R-Type	R[$rd] ← R[$rs] - R[$rt]
and $rd, $rs, $rt	R-Type	R[$rd] ← R[$rs] & R[$rt]
or $rd, $rs, $rt	R-Type	R[$rd] ← R[$rs] | R[$rt]
nor $rd, $rs, $rt	R-Type	R[$rd] ← ~ (R[$rs] | R[$rt])
slt $rd, $rs, $rt	R-Type	R[$rd] ← (R[$rs] < R[$rt])
slti $rt, $rs, imm	I-Type	R[$rt] ← (R[$rs] < SignExt(imm))
addi $rt, $rs, imm	I-Type	R[$rt] ← R[$rs] + SignExt(imm)
andi $rt, $rs, imm	I-Type	R[$rt] ← R[$rs] & ZeroExt(imm)
ori $rt, $rs, imm	I-Type	R[$rt] ← R[$rs] | ZeroExt(imm)
lw $rt, imm($rs)	I-Type	R[$rt] ← Mem[R[$rs] + SignExt(imm)]
sw $rt, imm($rs)	I-Type	Mem[R[$rs] + SignExt(imm)] ← R[$rt]
beq $rs, $rt, imm	I-Type	if (R[$rs] == R[$rt]) PC ← PC + 1 + (SignExt(imm))
j address	J-Type	PC ← {PC[31:26], address}
jal address	J-Type	R[31] ← PC + 1 PC ← {PC[31:26], address}
mult $rs, $rt	R-Type	{HI, LO} ← R[$rs] * R[$rt]
div $rs, $rt	R-Type	LO ← R[$rs] / R[$rt] HI ← R[$rs] % R[$rt]
jr $rs	R-Type	PC ← R[$rs]
mfhi $rd	R-Type	R[$rd] ← HI
mflo $rd	R-Type	R[$rd] ← LO


DECODER_CONTROL
decoder
ALUSrc
ALU_Control[3:0]
Branch
Jump
MemtoReg[1:0]
RegDst
decoder_done
imm_extended[31:0]
jump_address[25:0]
path_index[3:0]
rd[4:0]
rs[4:0]
rt[4:0]
select_shamt
shamt[4:0]
clk
en
instr[31:0]
decoder_control
BRAM MODULE
mem
addr[15:0]
clk
din[31:0]
en
ren
wen
dout[31:0]
MemReadWrite
JUMP MODULE
jump
addr[25:0]
clk
en
jump
path_index[3:0]
pc[31:0]
reg_addr[31:0]
jump_done
pc_out[31:0]
JumpModule
BRANCH MODULE
branch
alu_zero
branch
clk
en
imm[31:0]
pc[31:0]
branch_done
pc_out[31:0]
BranchModule
alu_srcA_MUX
alu_srcA_mux
read_data1[31:0]
select_shamt
shamt[4:0]
alu_srcA[31:0]
ALU_SRC_MUX
MUX 4x1 RegFile
write_reg_mux
SEL0_RegDst
SEL1_Jump
fix_ra[4:0]
rd[4:0]
rt[4:0]
mux_out[4:0]
MUX4x2_RegFile
Register File
regs
clk
en
read_reg1[4:0]
read_reg2[4:0]
reg_write
write_data[31:0]
write_reg[4:0]
read_data1[31:0]
read_data2[31:0]
register_done
RegisterFile
2x1 MUX
alu_srcB_mux
SEL
input0[31:0]
input1[31:0]
mux_out[31:0]
MUX2x1
ALU
alu
alu_control[3:0]
alu_srcA[31:0]
alu_srcB[31:0]
clk
en
alu_done
alu_result[31:0]
alu_zero
hi[31:0]
lo[31:0]
ALU
MUX 8x1 RegWrite
write_reg_data_mux
PC_updated[31:0]
SEL0_MemtoReg0
SEL1_MemtoReg1
SEL2_Jump
alu_result[31:0]
dout[31:0]
hi[31:0]
lo[31:0]
mux_out[31:0]
MUX8x3_RegWrite

BRAM Setup

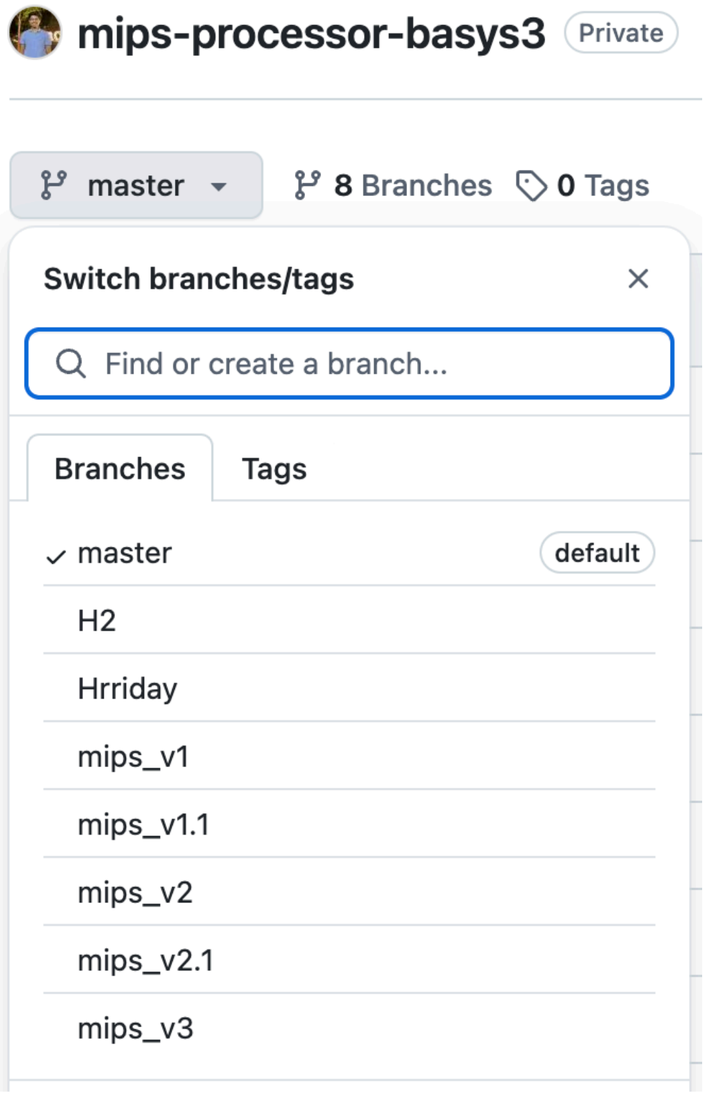

MIPS32
32
Word Indices
Byte Addr
(7fff fffc)
stack
Dynamic Data
Static Data
Text
Reserved
(1000 8000)
(1000 0000)
(0040 0000)
(0000 0000)
Hex
$2^{30}-1$
$2^{27}+2^{14}$
2^{27}
2^{21}
0
87.49%
0.00152%
12.30%
0.19%
2^{30} words
32 width
Ours (BRAM) [Proposed]
32
Word Indices
stack
Dynamic Data
Static Data
Text
Reserved
51199
6400.78125
6400
100
0
Issue!!!
Remove Reserved
51200 words
32 width
Ours (BRAM) [Final]
32
Word Indices
stack
Dynamic Data
Static Data
Text
Reserved
51199
6400
6300
0
87.5%
0.2%
12.30%
51200 words
32 width

BRAM Read

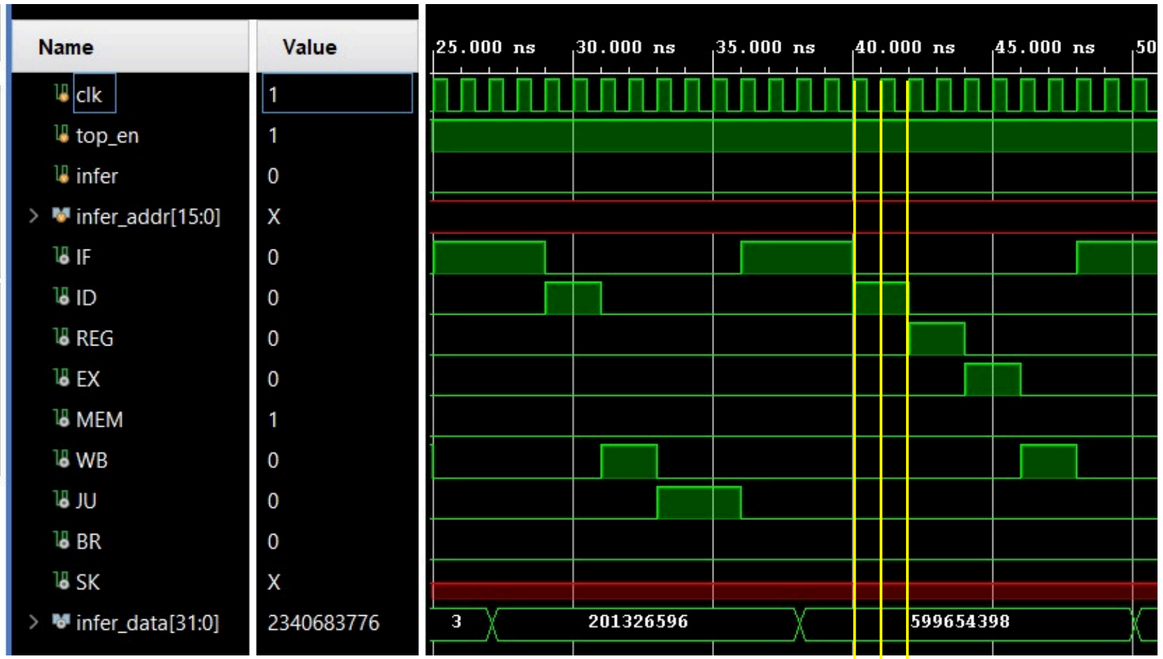

MemReadWrite.v
MemRW_tb.v
Untitled 1
Name
Value
clk
1
en
1
wen
0
addr[15:0]
c801
din[31:0]
00000001
dout[31:0]
0000c7ff
2cycles
2 cycles
2cycles
2 cycles
2cycles
57.000 ns
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 n
0000
0001
0002
0001
0005
c7ff
c801
0000
00000000
00000002
00000003
00000002
00000004
00000003
00000000
00000001
00000002
00000000
00000001
00000002
00000001
00000005
0000c7ff
XXXXXXXX
00000001
en=0
en=1
en=0
en=1
en=0
en=1
en=0
wen=1
en=1
en=0
en=1
en=0
en=1
en=0
en=1
wen=1
Inference:
Will need 2 extra redundant states after mem OP
If you see carefully, dout at this location in previously read value (even after 2 cycles)
Important:
en=0, wen=X ⇒ No mem OP
en=1, wen=0 ⇒ Read @ addr → dout
en=1, wen=1 ⇒ Write din @ addr

Versions of our Processor

1. **V1.0:** Uses done and complex sequential logic for feedback based FSM. Each module (except MEM) have clk and done signal. **Simulation-ready code**.
2. **V1.1:** Simple optimization. en on posedge and done checking on negedge (instead of posedge). **Simulation-ready code**.
3. **V2.0:** **Stable extension** of **mips_v1** for FPGA implementation . Factorial of 7 works on this.
4. **V2.1:** Unstable and incomplete extension of **mips_v1.1** for FPGA implementation. Multiple Driven nets issue at en of all modules. **Fix**: Add mux at each en that selects which clock edge will drive the en (sel = clk, 0 - negedge, 1 - posedge).
5. **V3.0:** Unstable extension of **mips_v2** . Kishan's attempt to remove done signals and simplify sequential logic. **Simulation not run.**
6. **V3.1:** **(Multi-State Processor - Each state takes exactly 1 clock cycle)** Stable extension of **mips_v2** with successful removal of done signals and simplification of modules to combinatorial blocks (withouth clk). Further optimized the memory read states and reduced the overall clock cycles by 1 for memory read. But now, decode consumes 2 clock cycles.



Versions of our Processor

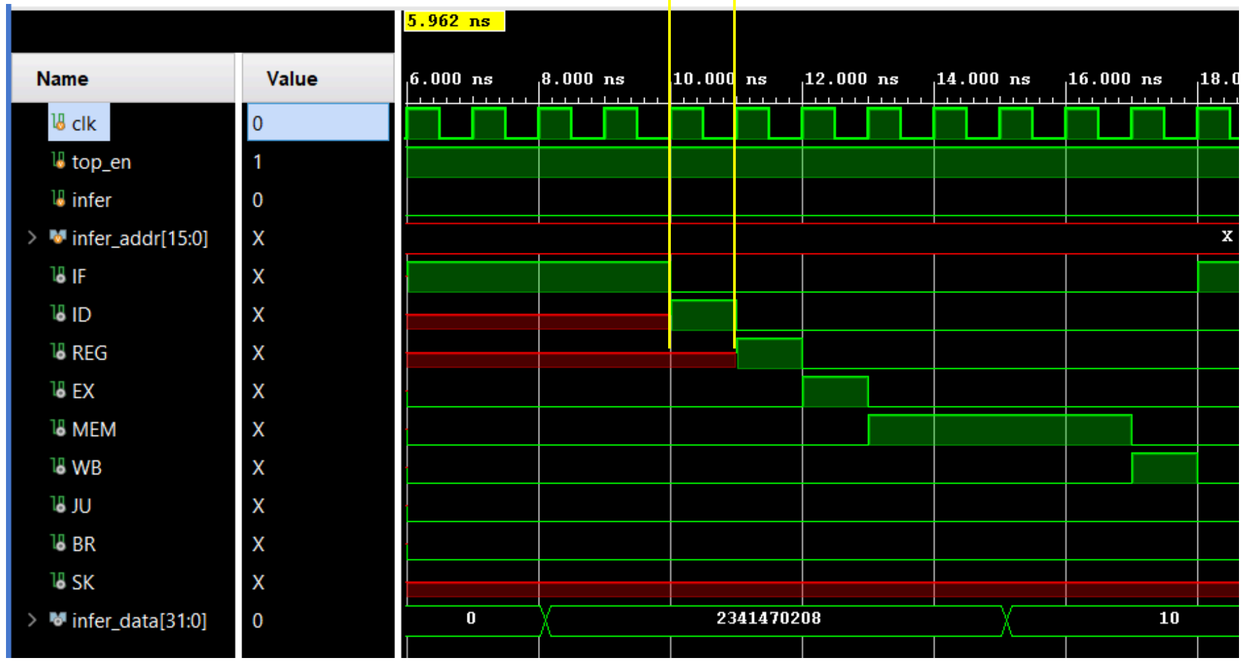


⟵mips_v1.0

Each State takes two CYCLES

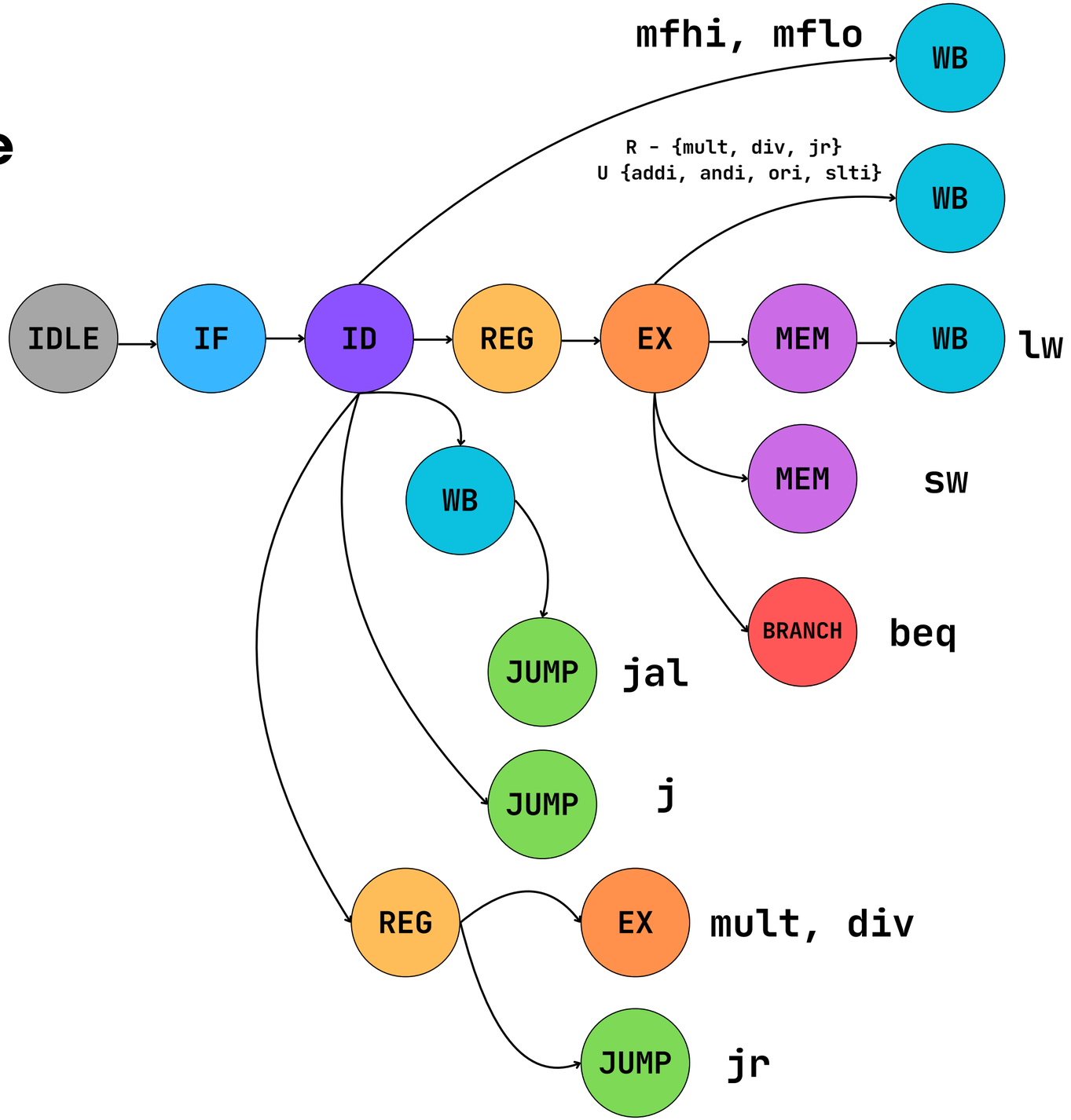
Each State takes one CYCLE

mips_v3.0⟶



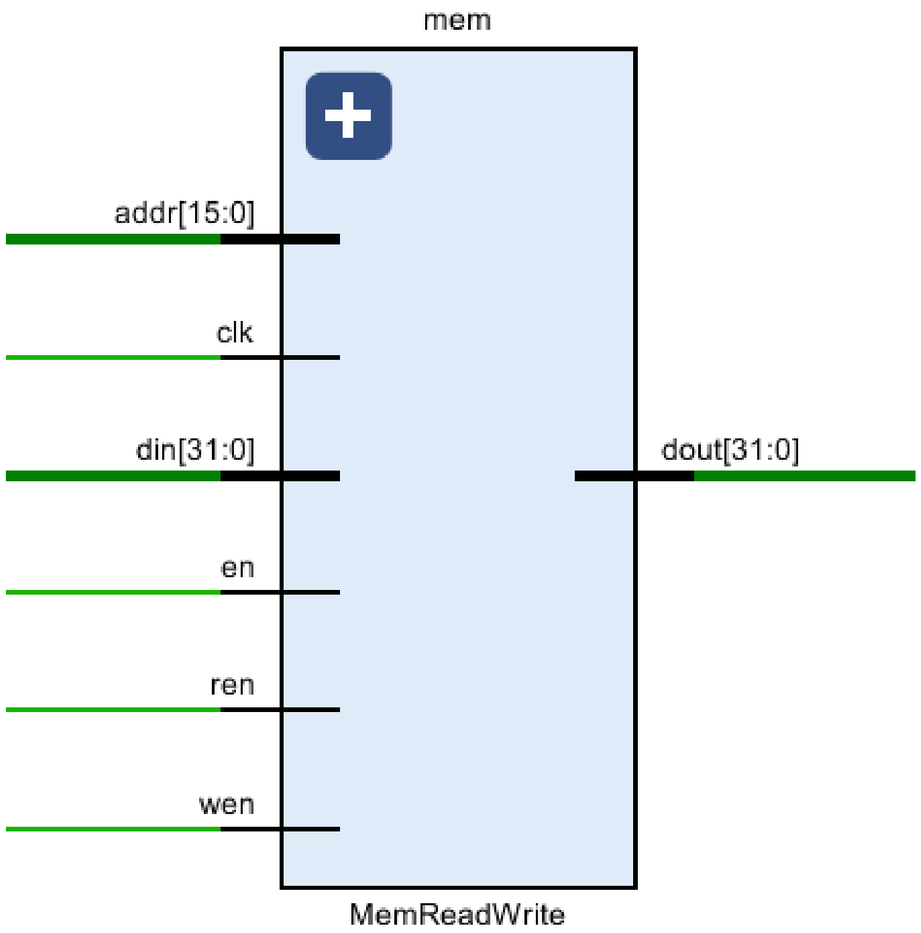
The State Machine & Path Index

mips_v2.0

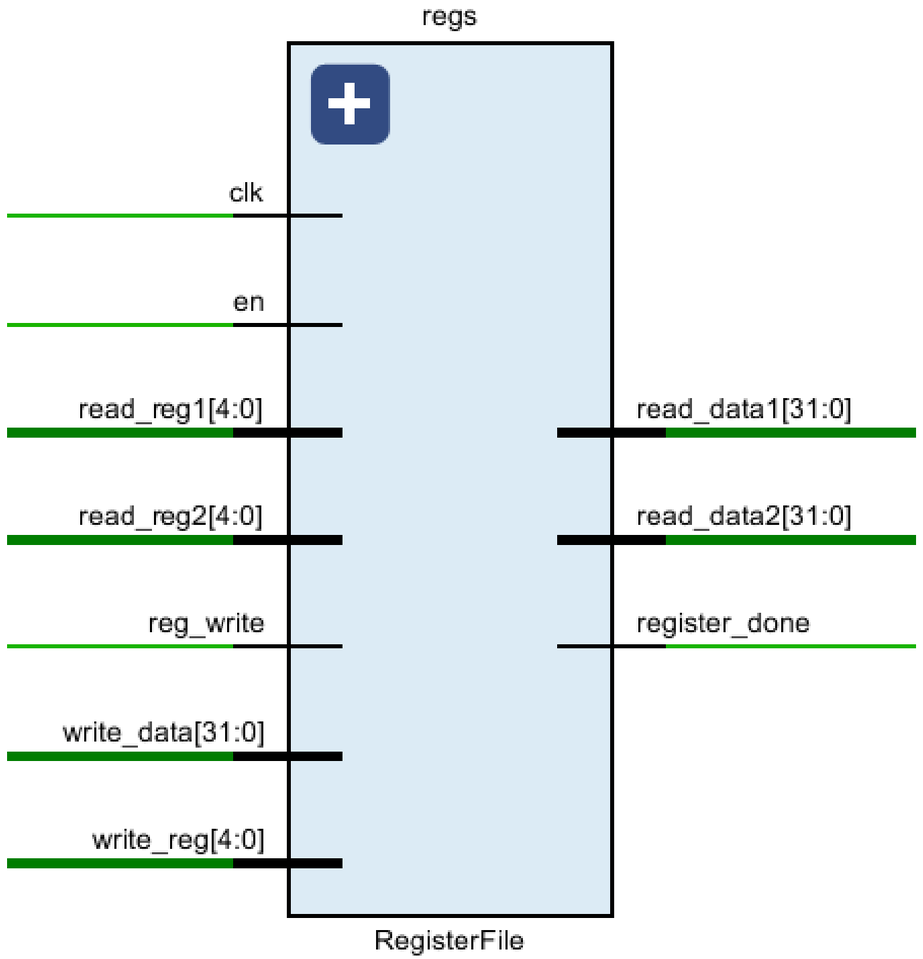


Modules

BRAM Memory Read-Write Module

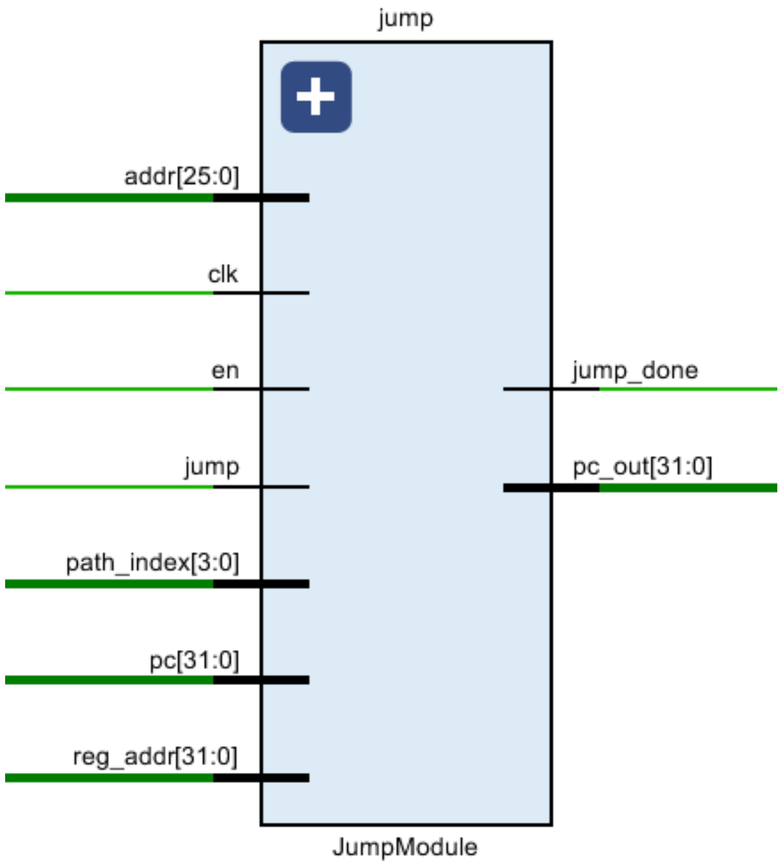


Register File

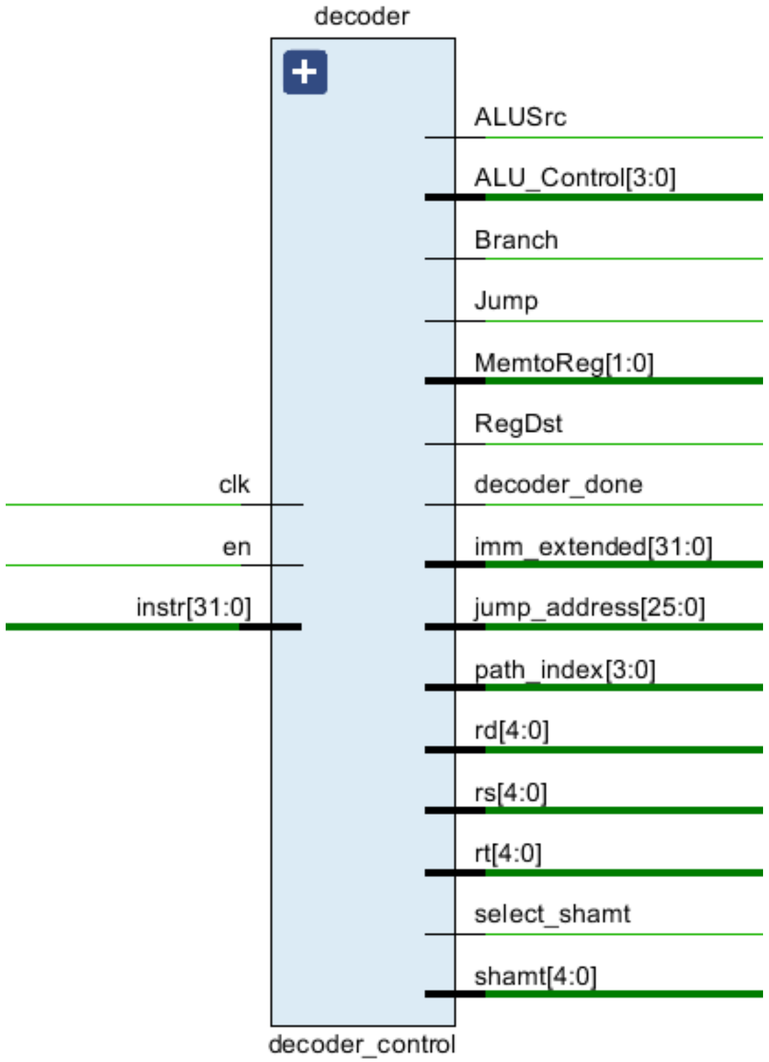


Modules

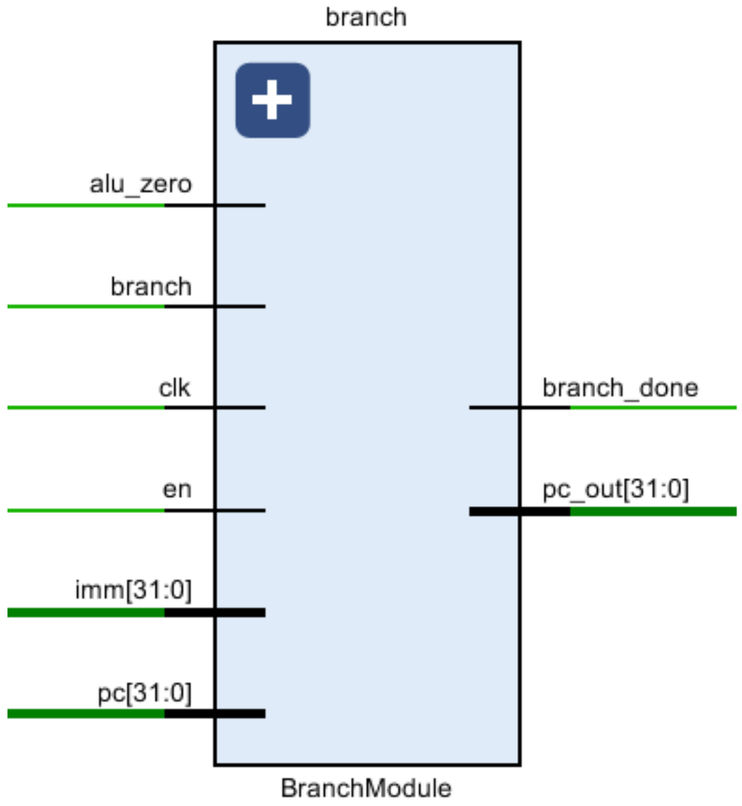
JUMP MODULE



DECODER-CONTROL MODULE

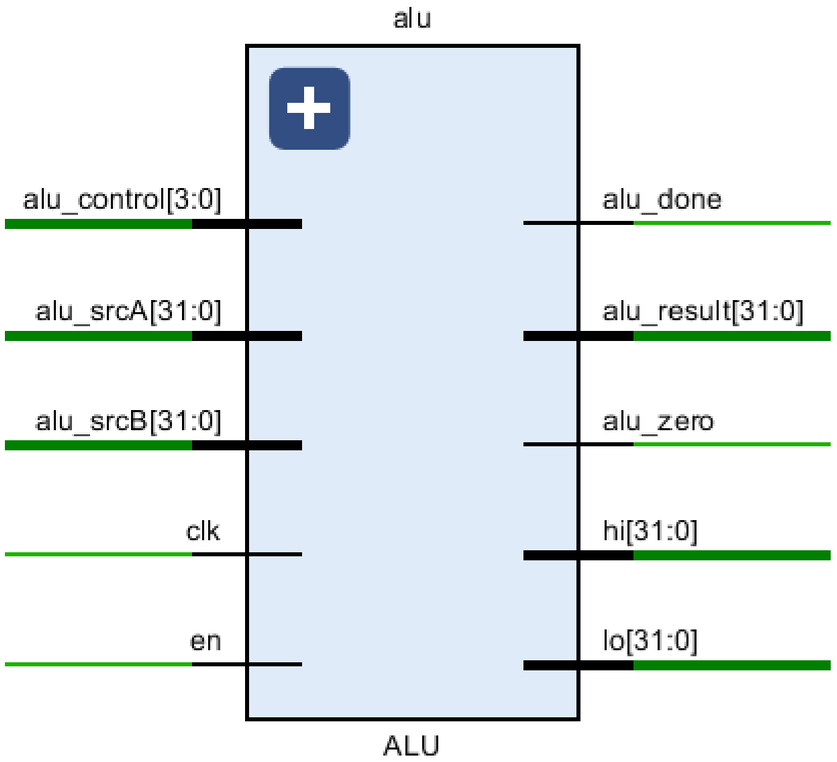


BRANCH MODULE



Modules

ALU




```
parameter ADD  = 4'b0000;
parameter SUB  = 4'b0001;
parameter AND  = 4'b0010;
parameter OR   = 4'b0011;
parameter NOR  = 4'b0100;
parameter SLT  = 4'b0101;
parameter SLL  = 4'b0110;
parameter SRL  = 4'b0111;
parameter MULT = 4'b1000;
parameter DIV  = 4'b1001;
```

MIPS Parser and COE Generator

```
main: lw $a0, 0($gp)
jal fact
sw $v0, 1($gp)
sysend
fact: addi $sp, $sp, -2
sw $ra, 1($sp)
sw $a0, 0($sp)
slti $t0, $a0, 1
beq $t0, $zero, L1
addi $v0, $zero, 1
addi $sp, $sp, 2
jr $ra
L1: addi $a0, $a0, -1
jal fact
lw $a0, 0($sp)
lw $ra, 1($sp)
addi $sp, $sp, 2
mult $a0, $v0
mflo $v0
jr $ra
```

mips_instructions.txt

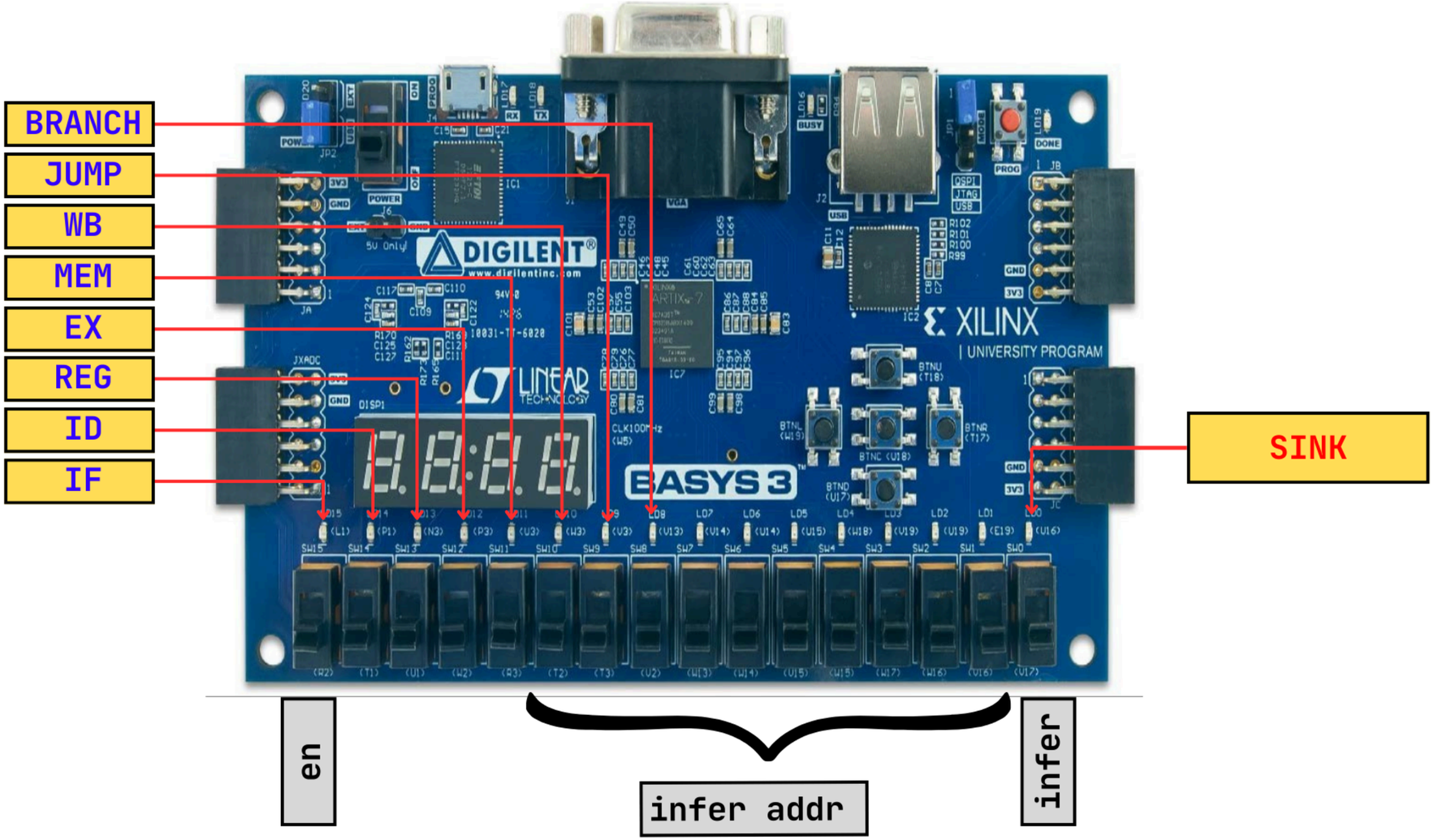
```
10001011100001000000000000000000
00001100000000000000000000000100
10101111100001000000000000000001
11111100000000000000000000000000
00100011101111011111111111111110
10101111101111110000000000000001
10101111101001000000000000000000
00101000100010000000000000000001
00010001000000000000000000000011
00100000000000100000000000000001
00100011101111010000000000000010
00000011111000000000000000001000
00100000100001001111111111111111
00001100000000000000000000000100
10001011101001000000000000000000
10001011101111110000000000000001
00100011101111010000000000000010
00000000100000100000000000011000
00000000000000000001000000010000
00000011111000000000000000001000
```

mips_machine_code.txt

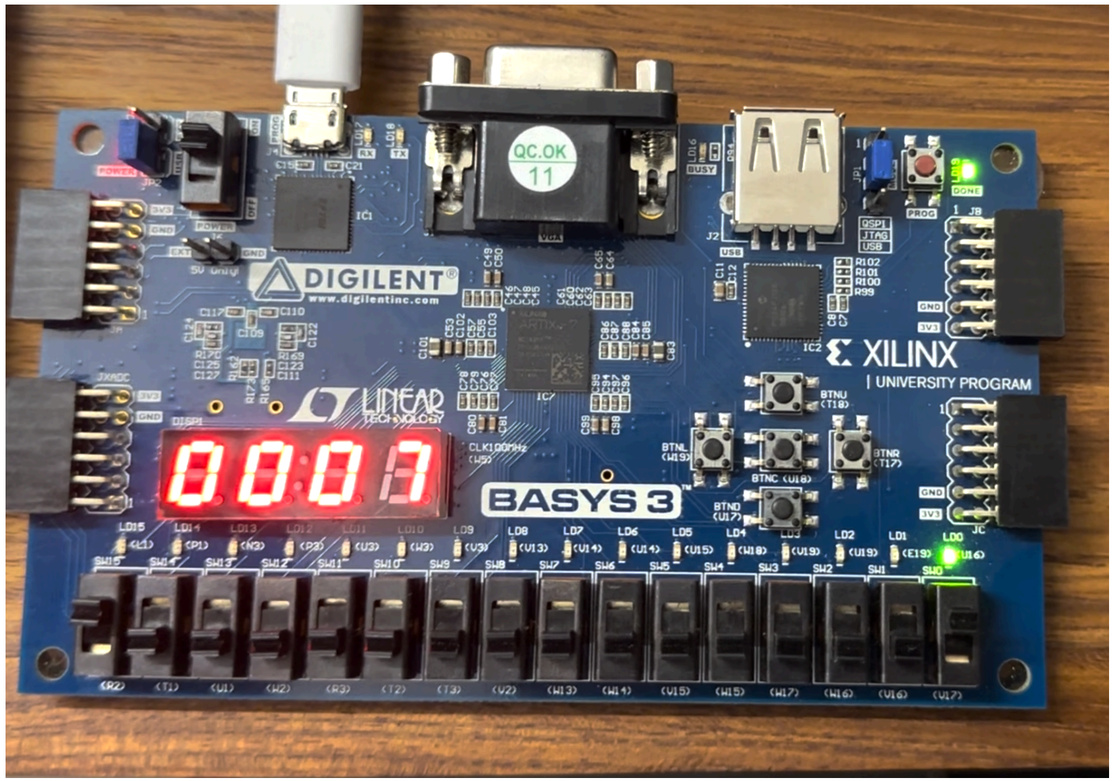
```
memory_initialization_radix=2;
memory_initialization_vector=
10001011100001000000000000000000,
00001100000000000000000000000100,
10101111100001000000000000000001,
11111100000000000000000000000000,
00100011101111011111111111111110,
10101111101111110000000000000001,
10101111101001000000000000000000,
00101000100010000000000000000001,
00010001000000000000000000000011,
00100000000000100000000000000001,
00100011101111010000000000000010,
00000011111000000000000000001000,
00100000100001001111111111111111,
00001100000000000000000000000100,
10001011101001000000000000000000,
10001011101111110000000000000001,
00100011101111010000000000000010,
00000000100000100000000000011000,
00000000000000000001000000010000,
00000011111000000000000000001000,
00000000000000000000000000000000,
00000000000000000000000000000000,
```

data.coe

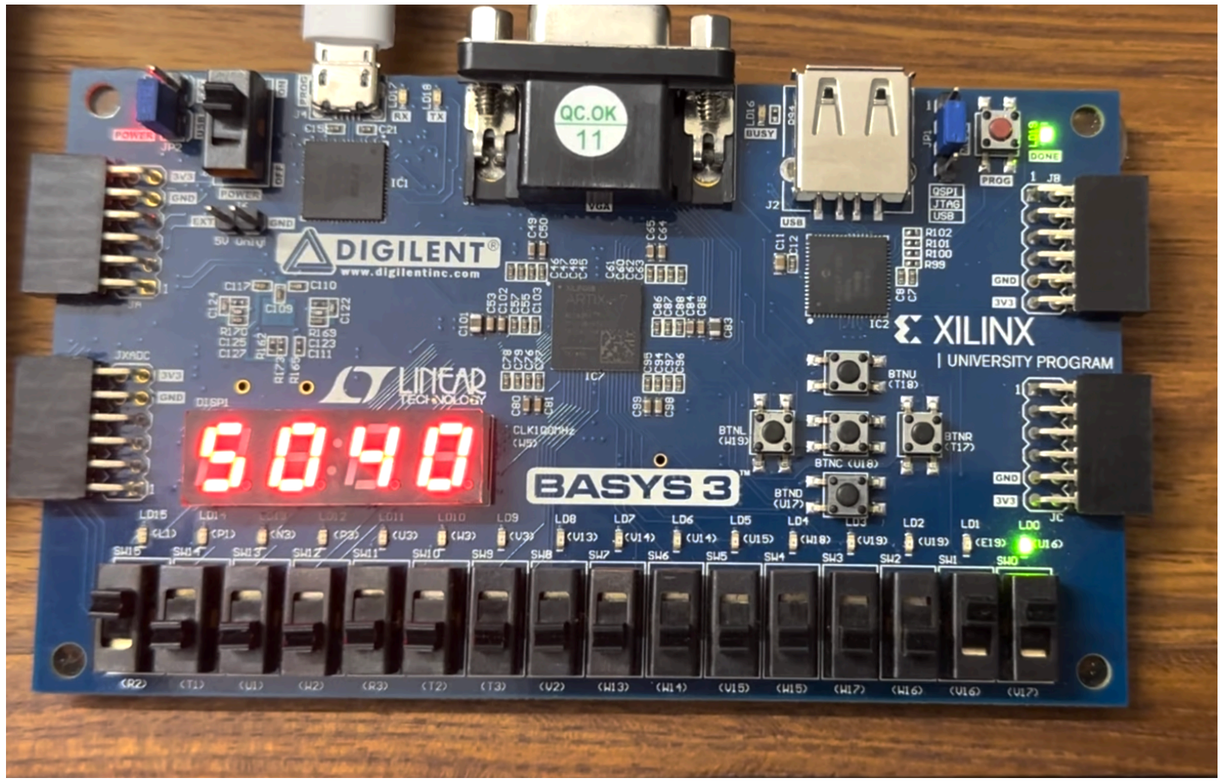
FPGA with Memory Mapped I/O and States Defined



Demonstration (Factorial)

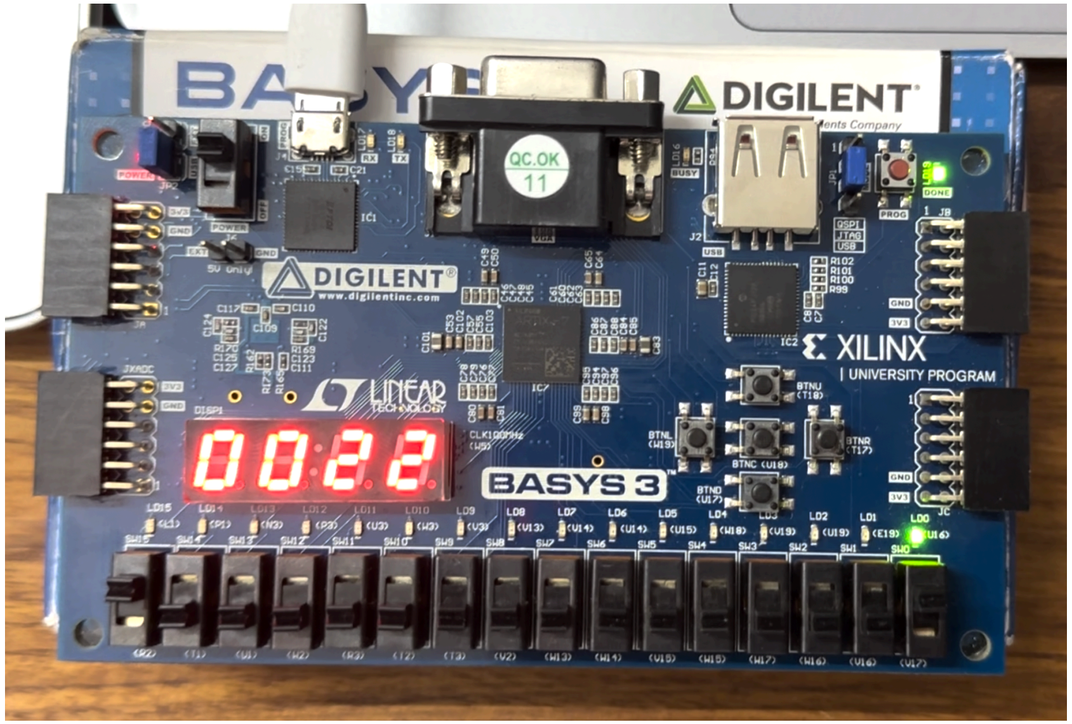


0($gp) -> 7

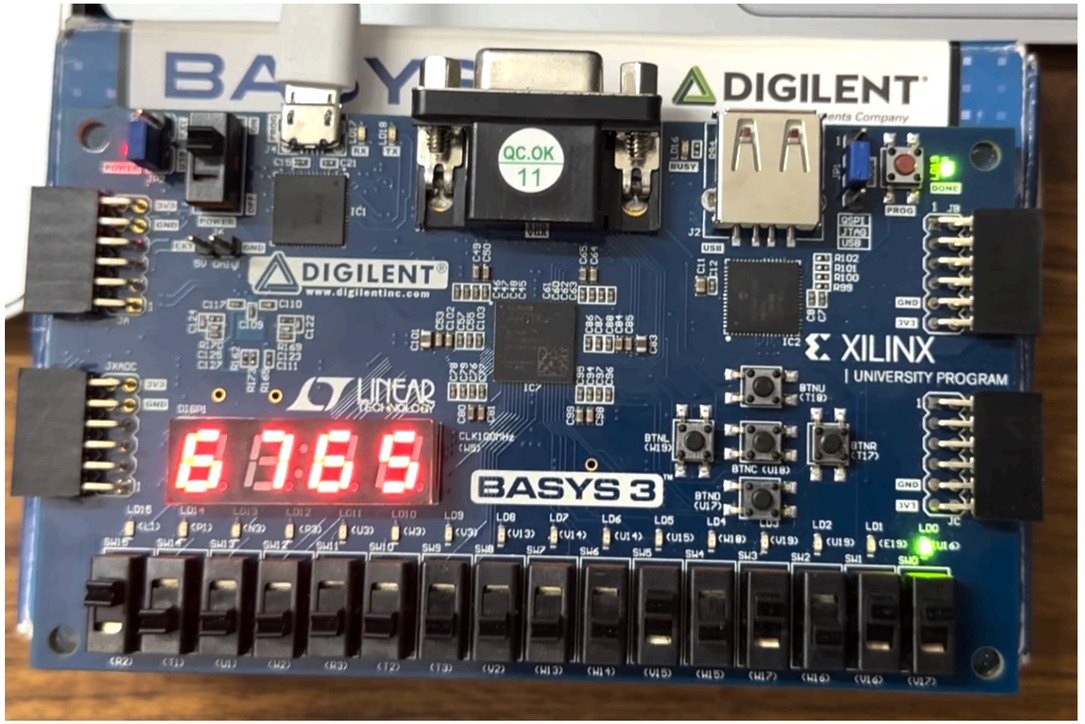


1($gp) -> 7! = 5040

Demonstration (Fibonacci)

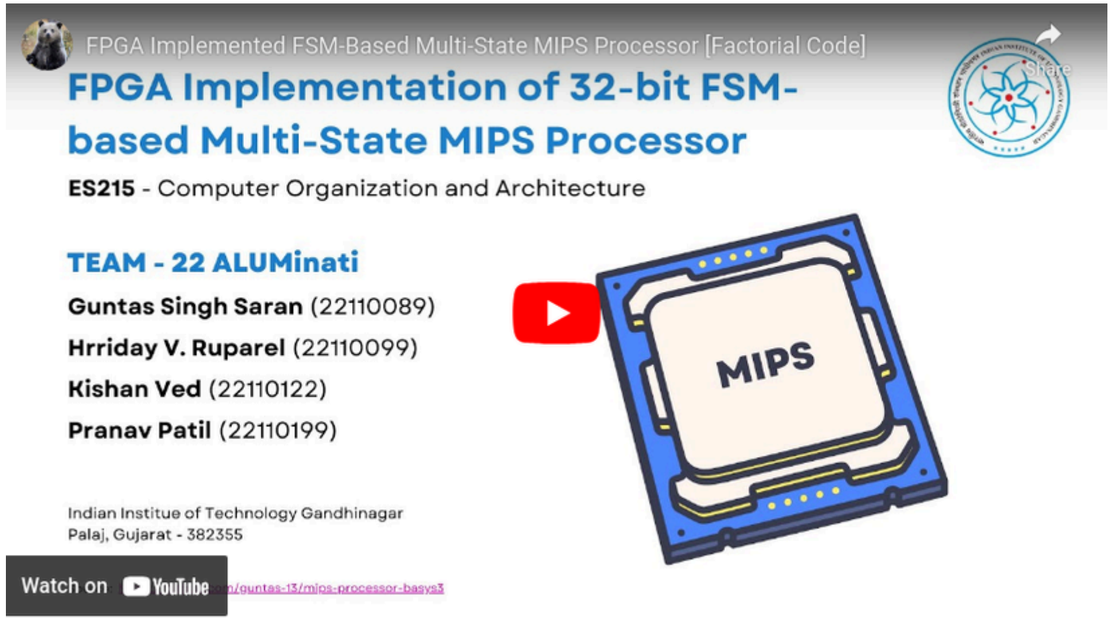


`0($gp) -> 22`

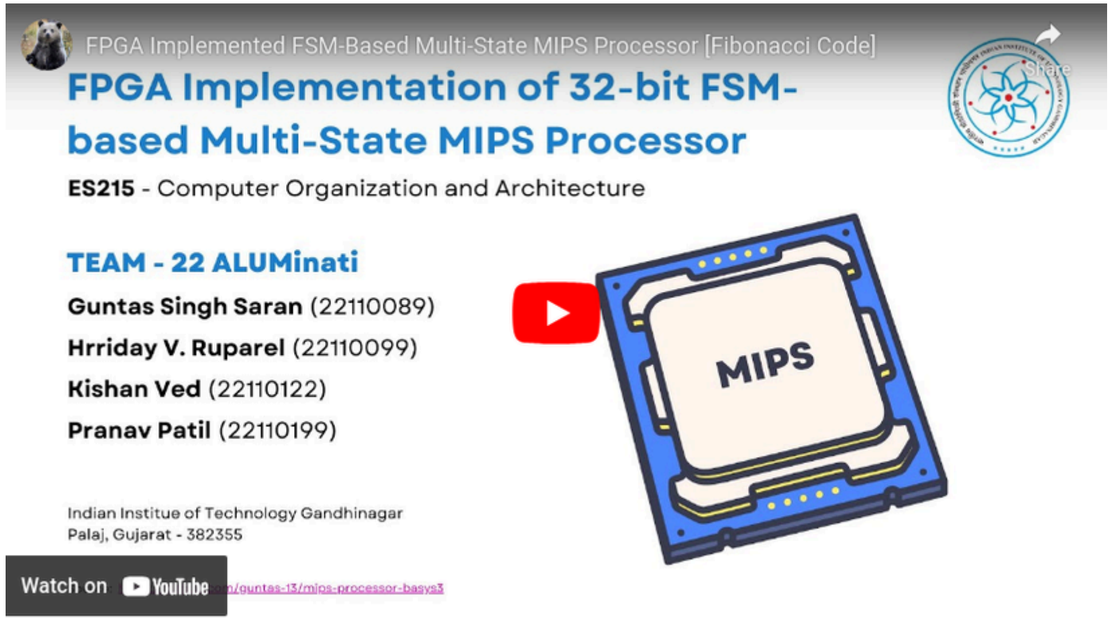


`21($gp) -> Fib(21) = 6765`

Demonstration Videos



https://www.youtube.com/watch?v=jQGQIGTQDpc



https://www.youtube.com/watch?v=N_pT4MgwUFg

THANK YOU

TEAM - 22 ALUminati

Guntas Singh Saran (guntassingh.saran@iitgn.ac.in)

Hrriday V. Ruparel (hrriday.ruparel@iitgn.ac.in)

Kishan Ved (kishan.ved@iitgn.ac.in)

Pravav Patil (pranav.patil@iitgn.ac.in)

Indian Institue of Technology Gandhinagar
Palaj, Gujarat - 382355