# CSE3015 / CSE3215 DIGITAL LOGIC DESIGN TERM PROJECT

In the last phase of the project, the complete processor datapath was integrated with a **Finite State Machine (FSM) based Control Unit**, resulting in a fully functional single-cycle multi-state processor implementation in Logisim. In this phase, all previously designed datapath components were connected and driven by control signals generated by the Control Unit according to the defined Instruction Set Architecture (ISA).

## 1. Overall Datapath Architecture

The complete datapath consists of the following main components:

- Program Counter (PC) – 12 bits
- Instruction Memory – 18-bit instruction width
- Register File – 16 registers (R0–R15), 18-bit data width
- Arithmetic Logic Unit (ALU)
- Extend Unit (Immediate / Address Extension)
- Data Memory – 12-bit address, 18-bit data
- Multiplexers for operand and PC selection
- Control Unit (FSM-based)

The datapath is designed in accordance with the fixed **18-bit instruction format** defined in the ISA. The opcode field (5 bits) is decoded by the Control Unit, while register indices, immediate values, and addresses are routed to the appropriate datapath components.

## 2. Control Unit Design Approach

The Control Unit is implemented as a **hardwired Finite State Machine (FSM)**, as required by the project specifications. Micro-programmed control was explicitly avoided. The FSM generates all necessary control signals for each clock cycle based on:

- Current instruction opcode
- Current FSM state

Each instruction is executed in multiple states, following a **fetch–decode–execute** style operation.

The Control Unit generates the following signals:

- **PCWrite** – Enables writing to the PC
- **PCSrc** – Selects PC+1 or PC+1+offset
- **RegWrite** – Enables register file write
- **MemRead** – Enables data memory read
- **MemWrite** – Enables data memory write
- **MemToReg** – Selects memory output for write-back
- **ALUSrc** – Selects ALU second operand (register or immediate)
- **ExtSel** – Selects sign or zero extension
- **ALUCtrl[2:0]** – Selects ALU operation

### 3. Instruction Fetch and Program Counter Control

PC output is connected to Instruction Memory address input. Instruction is fetched continuously from memory. During the fetch state:

- PC + 1 is computed.
- If the instruction is not a control-flow instruction, PC is updated with PC + 1.

For JUMP and JAL instructions:

- PC + 1 + offset is selected via a multiplexer.
- **PCSrc = 1** and **PCWrite = 1** update the PC.

### 4. ALU Operations and Operand Selection

The ALU supports the following operations as defined in the ISA:

- ADD, SUB
- NAND, NOR
- SRL, SRA

The ALUControl signal is a **3-bit control input**, allowing up to 8 different operations. Each instruction opcode is mapped to a unique ALUControl value.

Operand selection:

- First operand always comes from the Register File.
- Second operand is selected via **ALUSrc**:
    - 0 → Register value
    - 1 → Extended immediate

Immediate values are extended to 18 bits using the Extend Unit.

**5. Memory Access Instructions (LD / ST)**

For memory access instructions, the datapath operates as follows:

- **LD (Load)**:
    1. Address is extended and applied to Data Memory.
    2. MemRead = 1
    3. Memory output is selected using **MemToReg**
    4. **RegWrite = 1** writes data back to register
- **ST (Store)**:
    1. Register value is sent to Data Memory input.
    2. Address is extended.
    3. MemWrite = 1 enables memory write operation.

Only the lower 12 bits of the extended address are used.

**6. Stack Operations (PUSH / POP)**

Stack operations are implemented using **Register R15 as the Stack Pointer (SP)**.

**PUSH**

- Value is written to memory at address R15
- ALU computes R15 − 1
- Result is written back to R15

**POP**

- Value is read from memory at address R15

- ALU computes R15 + 1
- Result is written back to R15

**7. Control Flow Instructions (JUMP / JAL / CMOV)**

**JUMP / JAL**

- PC-relative offset is extended
- PC is updated using PC + 1 + offset
- JAL additionally writes PC + 1 to R15

**CMOV**

- ALU checks whether REG1 equals zero
- If true, REG3 is written into REG2
- No PC modification occurs

**8. Testing and Verification**

All datapath components and the Control Unit were integrated in Logisim. The processor was tested using custom assembly programs covering:

- Arithmetic and logic instructions
- Immediate operations
- Memory load/store
- Stack push/pop
- Control flow instructions

Assembly programs were converted into machine code using the custom Python assembler and loaded into Instruction Memory using Logisim's memory initialization feature. Correct execution was verified by observing register values, memory contents, and PC behavior during simulation.

**9. Summary**

In this final phase, a complete processor datapath with an FSM-based Control Unit was successfully designed and implemented. The design fully complies with the defined ISA and project constraints. The modular structure of the datapath and clear control signal separation ensure correctness, extensibility, and ease of debugging. The Logisim implementation

demonstrates that all instructions execute correctly across multiple states under the control of the finite state machine.

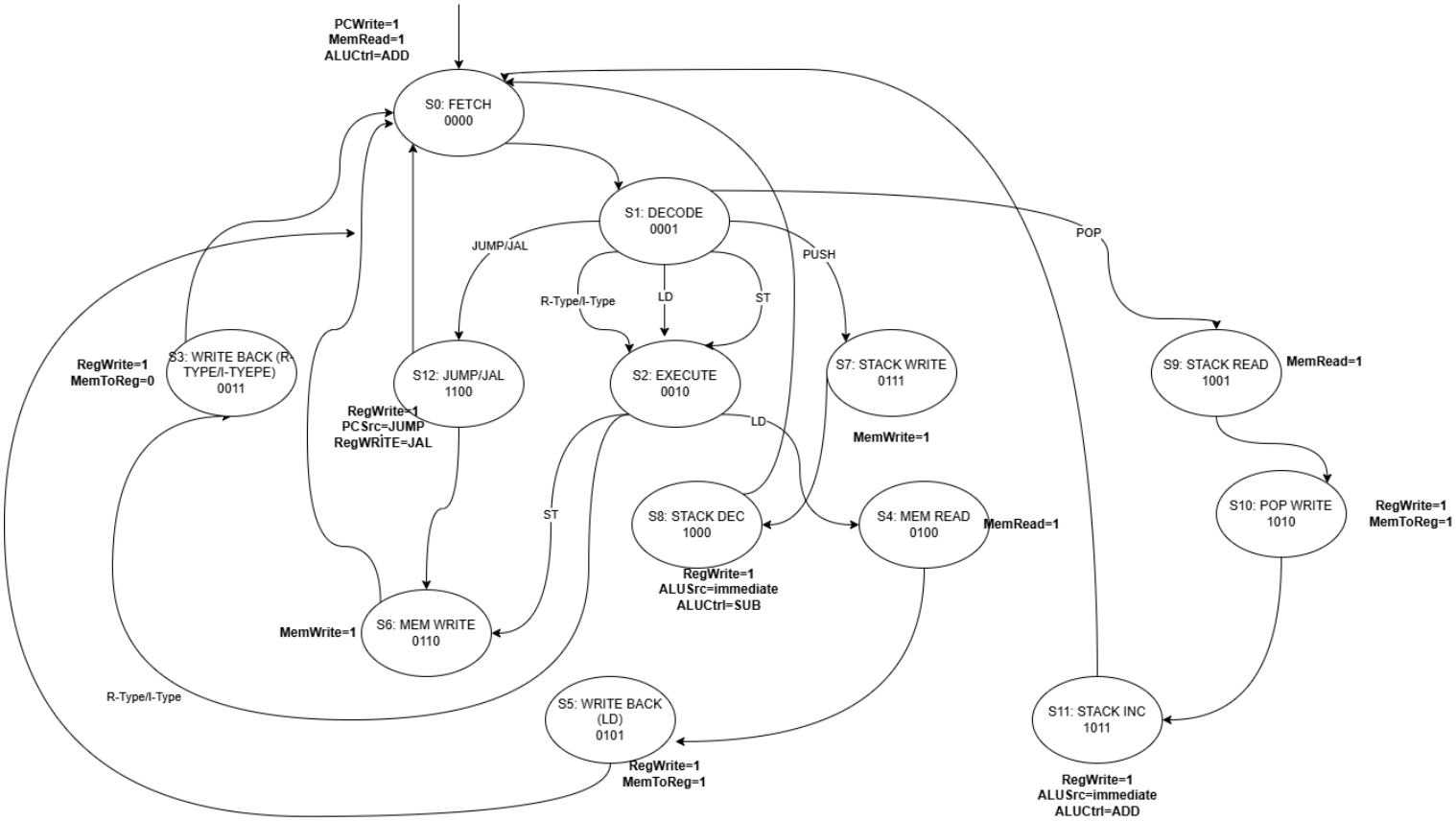


*Table 1:The finite state machine*

| Type | Format | Instruction | Opcode |
|---|---|---|---|
| R-Type | Opcode (5) + Dest (4) + Src1 (4) + Src2 (4) + Unused (1) | ADD | 00000 |
| | | SUB | 00001 |
| | | NAND | 00010 |
| | | NOR | 00011 |
| | | SRL | 00100 |
| | | SRA | 00101 |
| I-Type | Opcode (5) + Dest (4) + Src1 (4) + Immediate (5) | ADDI | 00110 |
| | | SUBI | 00111 |
| | | NANDI | 01000 |
| | | NORI | 01001 |
| J-Type | Opcode (5) + Address (13) | JUMP | 01010 |
| | | JAL | 01011 |
| Memory Access | Opcode (5) + Dest (4) + Address (9) | LD | 01100 |
| | Opcode (5) + Src (4) + Address (9) | ST | 01101 |
| Stack Operations | Opcode (5) + "0000" + Src1 (4) + "0000" + "0" | PUSH | 01110 |
| | Opcode (5) + Dest (4) + "0000" + "0000" + "0" | POP | 01111 |
| U-Type | Opcode (5) + Dest (4) + Immediate (9) | LUI | 10000 |
| R-Type | Opcode (5) + Reg1 (4) + Reg2 (4) + Reg3 (4) + Unused (1) | CMOV | 10001 |

*Table 2:The 18-bit Instruction Set Architecture Map*