

**Verilog Assignment 7 Part 4**  
**Instruction Format, Data Path, and Control Unit Design**  
**Group 27**  
**Sanskar Mittal- 21CS10057**  
**Yash Sirvi- 21CS10083**

**Question:**

Implement memory as a one-dimensional register array.

Complete the processor design by going through the following steps:

- a) Implement the data path using structural design methodology.
- b) Prepare a table depicting the (sequence of) RTL micro-operations corresponding to typical instructions like ADD, ADDI, LD, ST, BMI, MOVE, CALL and RET, along with the corresponding control signals required at every step.
- c) Implement the control path using behavioural design of the required FSM.
- d) Download the design on FPGA and test the functionality

**Overall processor format:**

32 bit instructions are fed to the processor which is processed instruction-by-instruction by the control unit.

The control unit generates the necessary logic to initialize different parts of the computation to be performed.

Results are stored in the register and can be stored in the main memory.

Similarly, data stored in the main memory can be fetched in a data register on which further operations can be performed.

**Verilog Implementation:**

We created different modules to represent different components of the processor architecture.

The main module **risc** interacts with the various other modules and acts as a bus. The **control unit** is responsible for processing the instructions in a step wise manner and generates appropriate signals for other components.

Operation	OPCODE (in binary)	Function Code (in binary)
ADD	000000	00000
SUB	000000	00001
AND	000000	00010
OR	000000	00011
XOR	000000	00100
NOT	000000	00101
SLA	000000	00110
SRA	000000	00111
SRL	000000	01000
ADDI	000001	NA
SUBI	000101	NA
SLAI	001001	NA
SRAI	001101	NA
SRLI	010001	NA
LD	010101	NA
ST	011001	NA
LDSP	011101	NA
STSP	100001	NA
MOVE	100101	NA
ANDI	101001	NA
ORI	101101	NA
XORI	110001	NA
BR	000010	NA
BMI	000110	NA
BPL	001010	NA
BZ	001110	NA
PUSH	010010	NA
POP	010110	NA
CALL	011010	NA
RET	011110	NA
HALT	100010	NA
NOP	100110	NA

Instruction Format Encoding-

- Last two bits of the opcode:
  - 00 -> ALU instructions
  - 01 -> I type
  - 10 -> J type (custom)

Register type Instructions-

OPCODE	Source Register 1	Source Register 2	Destination Register	Shamt	Func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Examples:

Code	OPCODE	Source Reg1	Source Reg2	Destination Reg	Shamt	Func
ADD R3, R1, R2	000000	00001	00010	00011	00000	000000
SUB R3, R1, R2	000000	00001	00010	00011	00000	000001
AND R3, R1, R2	000000	00001	00010	00011	00000	000010
OR R3, R1, R2	000000	00001	00010	00011	00000	000011
XOR R3, R1, R2	000000	00001	00010	00011	00000	000100
NOT R1, R2	000000	00010	00010	00001	00000	000101
SLA R1, R2	000000	00010	00000	00001	00000	000110
SRA R1, R2	000000	00010	00000	00001	00000	000111
SRL R1, R2	000000	00010	00000	00001	00000	001000

- First 6 bits for the opcode
- Next 5 bits are reserved for the first source register
- Next 5 bits are reserved for the second source register
- Next 5 bits are reserved for the destination register
- Next 5 bits are reserved for the shamt: shift amount (max 31)
- Next and the last 6 bits are reserved for the function name

Immediate Type Instructions-

OPCODE	Register 1	Register 2	Immediate Value
6 bits	5 bits	5 bits	16 bits

- First 6 bits are reserved for opcode
- Next 5 bits are reserved for first register
- Next 5 bits are reserved for the second register
- Next 16 bits are reserved for the immediate value

Examples:

Code	OPCODE	Register 1	Register 2	Immediate Value
ADDI R1, #2	000001	00001	00000	00000000000000010
SUBI R1, #2	000101	00001	00000	00000000000000010
SLAI R1, #1	001001	00001	00000	00000000000000001
SRAI R1, #1	001101	00001	00000	00000000000000001
SRLI R1, #1	010001	00001	00000	00000000000000001
LD R1, 10(R2)	010101	00010	00001	00000000000001010
ST R1, 2(R2)	011001	00001	00001	00000000000000010
LDSP SP, 0(R1)	011101	11101	00001	00000000000000000
STSP SP, 0(R1)	100001	11101	00001	00000000000000000
MOVE R1, R2	100101	00001	00010	00000000000000000
BR #10	000010	00000	00000	00000000000001010
BMI R1, #-10	000110	00001	00000	1111111111110110
BPL R1, #4	001010	00001	00000	00000000000000100
BZ R1, #8	001110	00001	00000	00000000000001000
PUSH R1	010010	00001	00000	00000000000000000
POP R1	010110	00001	00000	00000000000000000
CALL #2	010010	11101	00000	00000000000000010
RET	011110	11101	00000	00000000000000000

## Control unit-

It takes 6 bit opcode input and assigns value to different control lines.

- 0001 -> ADD
- 0010 -> SUB
- 0011 -> AND
- 0100 -> OR
- 0101 -> XOR
- 0110 -> NOT
- 0111-> SLA
- 1000 -> SRA
- 1001 -> SRL

CONTROL SIGNALS FOR ALU OPERATIONS WITH FUNC (R Type)

Opcode	Func	ALUop	ALUcontrol _input	ALUsource	Write Reg	MemWrite	MemRead*	MemRegPC	Stack Op	Branch
000000	000000	001	0001	1	1	0	0	0	00	000
000000	000001	001	0010	1	1	0	0	0	00	000
000000	000010	001	0011	1	1	0	0	0	00	000
000000	000011	001	0100	1	1	0	0	0	00	000
000000	000100	001	0101	1	1	0	0	0	00	000
000000	000101	001	0110	1	1	0	0	0	00	000
000000	000110	001	0111	1	1	0	0	0	00	000
000000	000111	001	1000	1	1	0	0	0	00	000
000000	001000	001	1001	1	1	0	0	0	00	000

### CONTROL SIGNALS FOR OPERATIONS WITHOUT FUNC (I Type)

Opcode	ALUOp	ALUcontrol_input	ALUsource	WriteReg	MemWrite	MemRead*	MemRegPC	Stack Op	Branch
000001	010	0001	0	1	0	0	0	00	000
000101	011	0010	0	1	0	0	0	00	000
001001	100	0011	0	1	0	0	0	00	000
001101	101	0100	0	1	0	0	0	00	000
010001	110	0101	0	1	0	0	0	00	000
010101	000	0001	0	1	0	1	1	00	000
011001	000	0001	0	0	1	1	0	00	000
011101	000	0001	0	1	0	1	1	00	000
100001	000	0001	0	0	1	0	0	00	000
000010	000	0001	0	0	0	0	0	00	001
000110	000	0001	0	0	0	0	0	00	011
001010	000	0001	0	0	0	0	0	00	101
001110	000	0001	0	0	0	0	0	00	111
010010	000	0000	0	0	1	1	0	10	000
010110	000	0000	0	1	0	1	0	01	000
011010	000	0000	0	0	1	1	0	11	001
011110	000	0000	0	1	0	1	1	01	000
100010	000	0000	0	0	0	0	0	00	000
100110	000	0000	0	0	0	0	0	00	000

- **Opcode:** Used to decide what type of instructions to generate
- **ALUOp:** This along with Func helps to generate ALUcontrol\_input
- **ALUcontrol\_input:** This tells ALU what ALU operation to perform
- **ALUsource:** Select Read Register 2 or Immediate Value
- **WriteReg:** 1 denotes write to register bank. 0 denotes .
- **MemWrite:** Write to memory or not (1/0)
- **MemRead:** Read from memory or not (1/0)
- **MemRegPC:** Select what to write to the Destination Register
  - 0 -> AluOut
  - 1 -> Memory output
- **Stackop:** Denotes what to do with stack pointer.
  - 00-> nothing.
  - 01-> increase SP by 4.
  - 11-> decrease SP by 4
- **Branch:** Denotes if branch is enabled or not. Branch[0] denotes no branch operation.
  - 001 -> Add immediate value to PC
  - 011 -> Add immediate value to PC if flag is set to 01
  - 101 -> Add immediate value to PC if flag is set to 10
  - 111 -> Add immediate value to PC if flag is set to 11

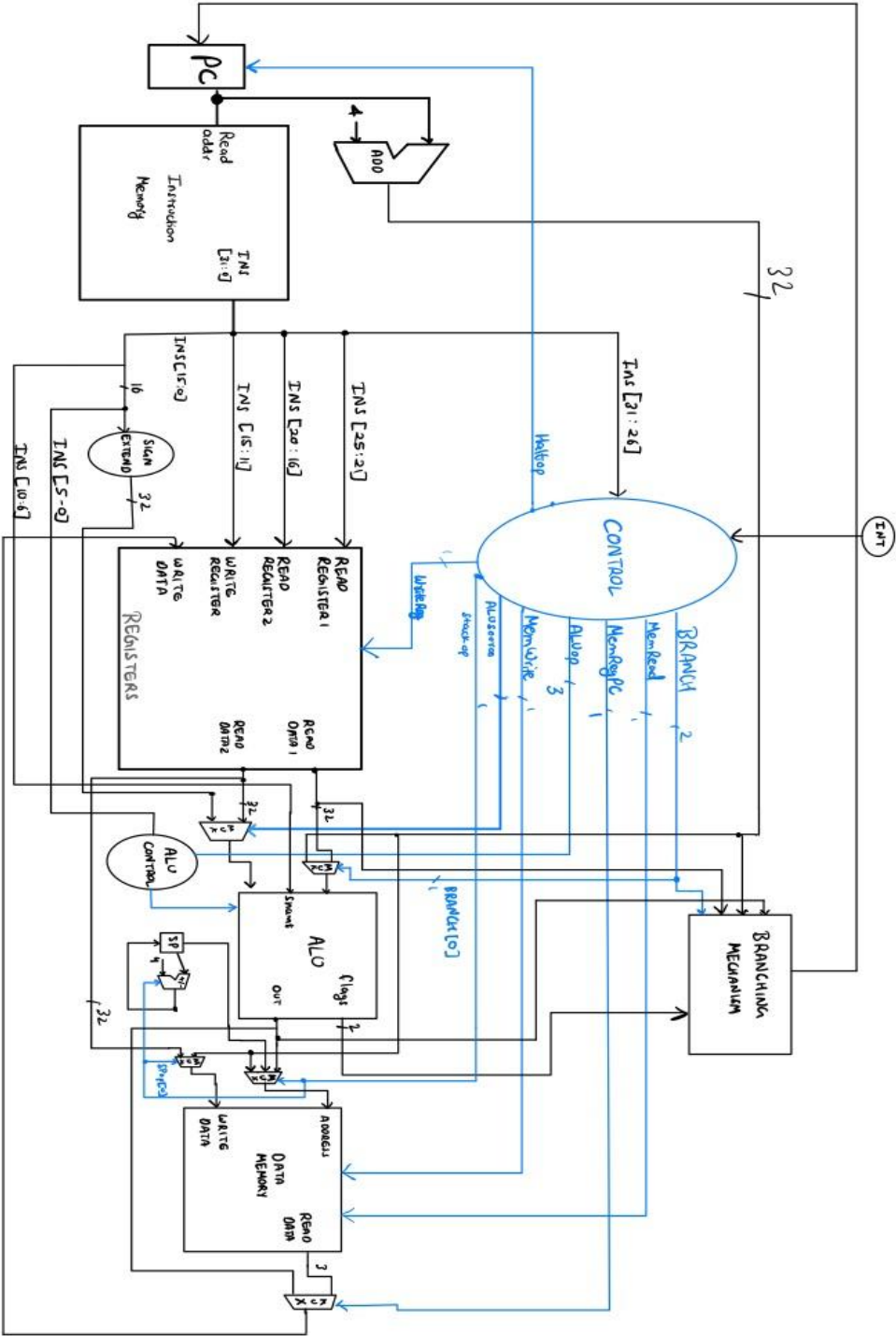
- \* Flag is the ALU output which tells whether the register value is < 0 (set to 01)
- > 0 (set to 10)
- = 0 (set to 11)

Halting Operation-

Opcode	HALTop	INT
100010	1	X
XXXXXX	0	Interrupt
Any other Opcode	0	X

- If the opcode corresponds to the HALT operation, we set HALTop to 1 which pauses Instruction Fetch and Decoding.
- When we receive an interrupt on the external pin **INT**, we reset HALTop to 0 which results in a normal Instruction execution cycle.

Control Unit Schematic Diagram





MICRO-OPERATIONS and Required CONTROL SIGNALS OF  
INSTRUCTIONS

imm <- immediate value in the instruction

ADD

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- Reg[Rt]	aluOp = 000 aluSource = 1
T2	C <- A + B	writeReg = 1
T3		writeReg = 0

SUBI

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- imm	aluOp = 010
T2	C <- A - B	writeReg = 1
T3		writeReg = 0

SLAI

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- imm	aluOp = 100
T2	C <- A <<< imm	writeReg = 1
T3		writeReg = 0

SRAI

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- imm	aluOp = 101
T2	C <- A >>> imm	writeReg = 1
T3		writeReg = 0

SRLI

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- imm	aluOp = 110
T2	C <- A >> B	writeReg = 1
T3		writeReg = 0

LD

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- imm	aluOp = 000 haltOp = 1
T2	temp <- Mem[A+B]	memRead = 1
T3	C <- temp	writeReg = 1 memRegPC = 1
T4		writeReg = 0

ST

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs] B <- imm	aluOp = 000 haltOp = 1 writeReg = 1
T2	Mem[A+B] <- C	memRead = 1 memWrite = 1
T3		writeReg = 0 memRegPC = 0
T4		memRead = 0 memWrite = 0

BR

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- imm	aluOp = 000 haltOp = 1
T2	PC <- PC + A	haltOp = 0
T3		haltOp = 1

PUSH

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs]	aluOp = 000 stackOp = 2 haltOp = 1
T2		// To complete prev step computation
T3	Stack.push(A)	memWrite = 1
T4		haltOp = 0

POP

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- Reg[Rs]	aluOp = 000 stackOp = 1 haltOp = 1
T2		memRead = 1
T3	A <- Stack.pop	memRead = 0 writeReg = 1
T4		writeReg = 0 memRegPC = 0
T5		writeReg = 1
T6		writeReg = 0 haltOp = 0

CALL

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- imm	aluOp = 000 stackOp = 2 haltOp = 1 memRead = 1
T2	SP <- SP - 4	writeReg = 1
T3	Mem[SP] <- PC + 4	writeReg = 0 memWrite = 1
T4	PC <- PC + imm	memWrite = 0
T5		haltOp = 0

RET

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1	A <- imm	aluOp = 000 memRegPC = 1 stackOp = 1
T2	Mem[PC] <- PC	memRead = 1
T3	SP <- SP + 4	memRead = 0 writeReg = 1
T4		writeReg = 0

HALT

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1		aluOp = 000 haltOp = 1
T2		haltOp = interrupt ? 0 : 1

NOP

TIMESTAMPS	RTL MICROOPERATIONS	CONTROL SIGNALS
T1		aluOp = 000

