Project Overview

What do you need to design

- 9-bit Processor:

  Instruction Set Architecture

  (a) Instruction Set: Lab 1

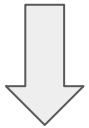
  (b) Architecture: Lab 2

- Program:

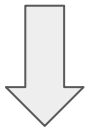
  Machine Code: Lab 3

Instruction Set (Lab 1)

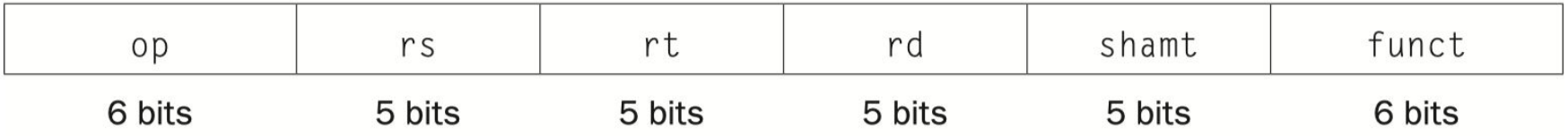
C code: A = B + C + D;



Assembly (MIPS):
add $t0, $s1, $s2
add $s0, $t0, $s3



Machine Code:
00000010001100100100000000100000
00000001000100111000000000100000



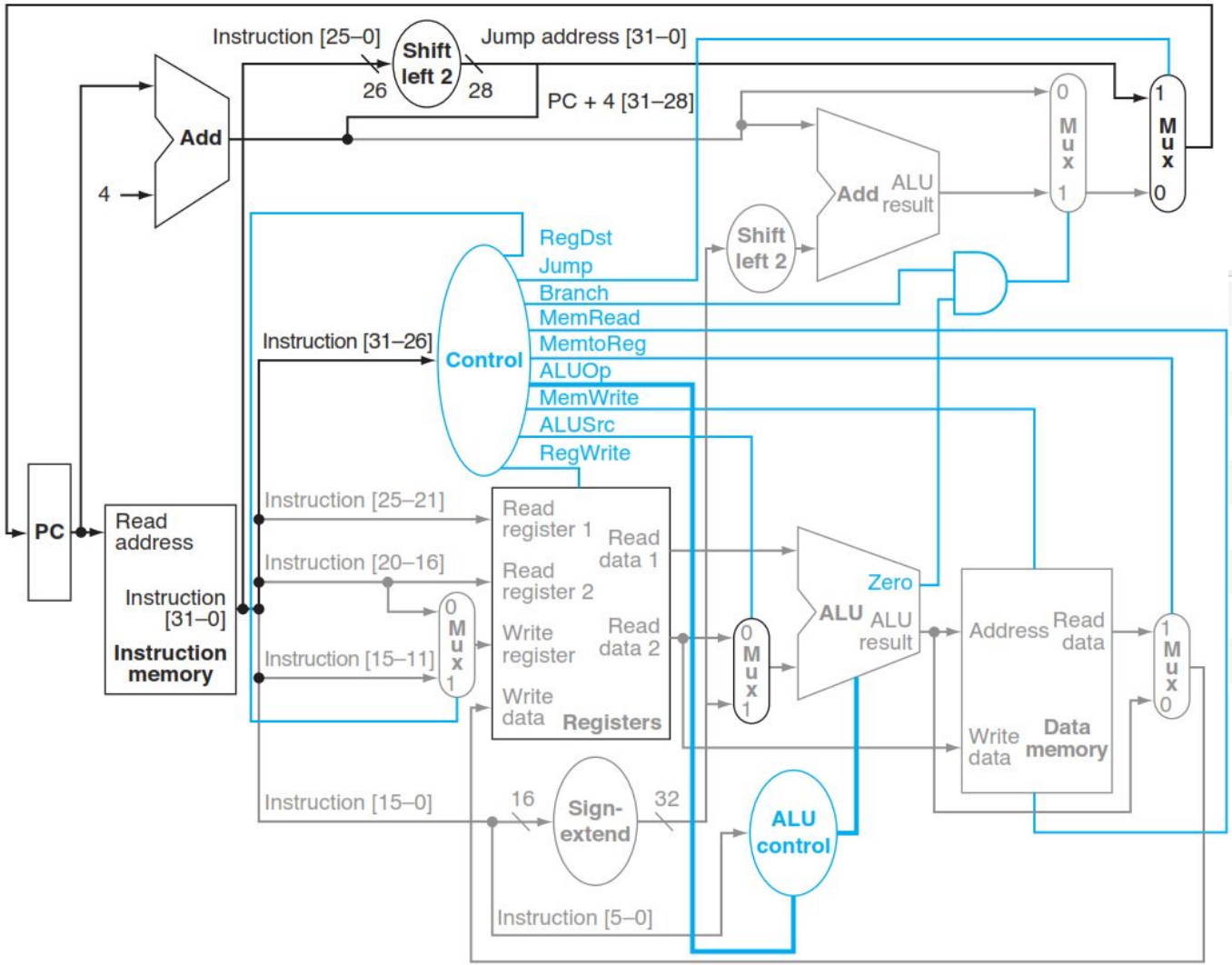
Here is the meaning of each name of the fields in MIPS instructions:

- *op:* Basic operation of the instruction, traditionally called the **opcode**.
- *rs:* The first register source operand.
- *rt:* The second register source operand.
- *rd:* The register destination operand. It gets the result of the operation.
- *shamt:* Shift amount. (Section 2.6 explains shift instructions and this term; it will not be used until then, and hence the field contains zero in this section.)
- *funct:* Function. This field, often called the *function code,* selects the specific variant of the operation in the op field.

Different types of ISA

Style	***# Operands***	***Example***	***Operation***
Stack	0	add	$tos_{(N-1)} \leftarrow tos_{(N)} + tos_{(N-1)}$
Accumulator	1	add A	acc ← acc + mem[A]
General Purpose	3	add A B Rc	mem[A] ← mem[B] + Rc
Register	2	add A Rc	mem[A] ← mem[A] + Rc
Load/Store:	3	add Ra Rb Rc	Ra ← Rb + Rc
		load Ra Rb	Ra ← mem[Rb]
		store Ra A	mem[A] ← Ra

Architecture (Lab 2)



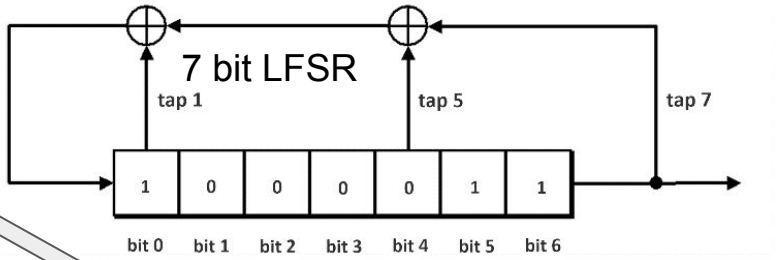
You might need LUT

- ALU.sv
- Ctrl.sv
- DataMem.sv
- Definitions.sv
- InstFetch.sv
- InstROM.sv
- LUT.sv
- RegFile.sv
- TopLevel.sv

Program 1: Encryption

256 Byte Data Memory, 1 Byte = 8 bit

DataMem[0: 48]: Original input Message	DataMem[61]: N of prepending space char	DataMem[62]: LFSR tap pattern **Index**. (Use the index to get the pattern)	DataMem[63]: LFSR Starting State	DataMem[64:64+N]: Encrypted Prepending Space	DataMem[64+N:127]: Encrypted Message + Encrypted Appending Space



Dest[i] = Source[i] ^ curr_LFSR
curr_LFSR = (curr_LFSR<<1)|(^(curr_LFSR & taps))

DataMem[64:128][7]: Parity Bit → ^[6:0]
DataMem[64:128][6:0]: Encrypted Message

LFSR Tap pattern table

```
assign LFSR_ptrn[0] = 7'h60;
assign LFSR_ptrn[1] = 7'h48;
assign LFSR_ptrn[2] = 7'h78;
assign LFSR_ptrn[3] = 7'h72;
assign LFSR_ptrn[4] = 7'h6A;
assign LFSR_ptrn[5] = 7'h69;
assign LFSR_ptrn[6] = 7'h5C;
assign LFSR_ptrn[7] = 7'h7E;
assign LFSR_ptrn[8] = 7'h7B;
```

Program 2: Decryption

DM[X] = DM[64+X] ^ curr_LFSR

DataMem[0:63]: Decrypted Message	DataMem[64:127]: Encrypted Message

1. Use DM[64] to calculate LFSR starting state
2. Loop through all 9 LFSR Tap pattern, do the decryption, and compare with DM[64:73], see which pattern is able to produce 10 ASCII space.
3. Do the decryption

Program 3: Upgraded Decryption

You will basically repeat Program 2 with some additional refinement

Difference with Program 2:

1. Remove all initial space, until a non space character. Then store the non space characters from DM[0](assuming no error)
2. Calculate the parity bit. Does DM[64 + X][7] == ^DM[64 + X][6:0]?

- If yes, then store the decrypted char into DM[X]
- If no, then store error flag 0x80 into DM[X]