Part 1: 3/2

Control unit Arch?

1. hardwired:
   - signals generated by fixed logic circuits
   - fast but inflexible (difficult to modify)
2. micro programmed:
   - control signals generated by microprogram
   - flexible, but more complex

SISC:

- Simple Instruction Set Computer
- multi-cycle RISC computer
- separates data and instruction memory

Word length: 32-bits
Bit-ordering: little endian
General purpose Registers: 16, 32-bit registers
Program Counter: 16 bits
Instruction/data space: 2^{16} bits → 64 kilo words (kew)
Addressing Resolution: 32 bits
Byte Ordering: little endian
Instruction set: LOAD/STORE (LSISC)
Register Operand Length: 32 bits
Immediate Operand Length: 16 bits
Clock Rate: 1 cycle per 10ns
cycles per Instruction: 5 (i.e. instructions take 50ns)

Control unit?

- part of CPU that manages execution of instructions by generating control signals
- coordinates operations on the data paths, memory, & other components

Function of control unit:
1. Instruction Fetch: retrieve instruction address in PC
2. Instruction Decode: decode instruction to determine operation & operands
3. Execution: generate signals to configure datapath (e.g. ALU operation, register selects)
4. Memory Access: load/store instruction requires control unit to manage memory access
5. Writeback: causes result to return to correct destination (register or memory)

Orchestrate | Manage resources | Determines Performance

operand vs Address?

- operand: * data or value operated on by instruction set; · specifies what the instruction should act on
- Address: * location in memory where data is stored or received; · specifies where data is stored

Ex: `ADD R1, R2` (operands) `MOVE R3, #5`

Ex: `LOAD R1, [0x1000]` (Address) `STORE R2, [R3]` (address contained in R3, pointer!)

Logic vs Control?

- Logic: * operations done on data; · ALU; Ex: adding two numbers
- Control: * coordination of operations & data movement; Ex: decoding instruction

Words?

- natural unit of data used by a particular computer architecture.
- * largest chunk of data that CPU can process in one operation
- + efficient: words align with CPU's natural data size
- + memory addressing: organized into word-sized chunks
- + Instruction Set Design: instructions designed to operate on words

Addressing Modes: Immediate, Register Indirect, Index, absolute, auto pre-increment, auto post-decrement

#value	(Ri)	X(Ri)	Loc	+(Ri)	(Ri)-
Operand = value	EA = [Ri]	EA = [Ri] + X	EA = Loc	EA incremented before	EA decremented after

Direct looks like this: `LOAD R1, [0x1000]` (known direct register location)

Spaces?

1. Von Neumann: shared instructions & data
   - x86, ARM
   - simplifies design; flexible
2. Harvard: separated instructions and data
   - PIC
   - improves performance; simultaneous instruction fetching & data accessing

GPRS?

* versatile registers
- why "general"?
  - not restricted to specific purpose
  - arithmetic, logic, addressing, or temporary storage → ADD, MOV, LOAD, etc
  - R0-R12 in ARM
  - EAX, EBX, ECX, EDX in x86

(unlike special purpose registers)

Special Registers: Program Counter (PC); Stack Pointer (SP); Instruction Register (IR)
Floating-Point Registers: store floating points
Vector Registers: store & process multiple values simultaneously
Status Register: flags (C, Z, V, N, etc)
Control Registers: manage CPU operations & configurations

Memory vs Registers?

RAM	Registers
· outside CPU	· inside CPU
· slow	· small & fast
· instructions	· limited
· data	· temp storage

Endians?

Not Indians

* specifies how bytes are stored in memory for multi-byte data types (int, float, etc)
* determines how data is Read & written to

Example: 0x12345678 (32 bit integer)

Little Endian
- LSB in lowest memory address →

address	0x1000	0x1001	0x1002	0x1003
data	0x78	0x56	0x34	0x12

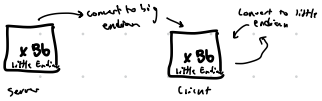
- 8086 x86 architecture, ARM

Big Endian
- MSB in lowest memory address →

address	0x1000	0x1001	0x1002	0x1003
data	0x12	0x34	0x56	0x78

- IBM network protocols (TCP/IP)

Networking:
- uses big endian



* Note, bits are always ordered: $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$
- Endians ARE FOR BYTES; not bits

Address Resolution?

- process of determining physical location in memory for corresponding memory address

Memory Address: unique identifier for a location in memory
Address Bus: hardware pathway used by CPU to specify an address bus
Addressable Memory: total amount of memory accessible by address bus
32 bit addresses → 2^{32} locations 4.3 GB 4 GB

Instruction set?

* collection of all instructions CPU can execute
- arithmetic, logic, data movement, control flow
- Binary code

ASM: human readable representation of instruction set. uses mnemonics

NOTE: hex is used as it is human readable.
Signals are BINARY

Addressing modes?

* how CPU interprets operands in an instruction to access data or memory
→ how the Effective Address (EA) is calculated by the CPU

Immediate, Register Indirect, Index.

- Immediate: loading constants or small values directly
- Register Indirect: accessing data via pointers or dynamic memory locations
- Index: accessing arrays; sequential data access

Bit Bucket?

used in typical conversation

* colloquial way to describe a register where data is written to, but not used or read back
* "throws away" contents placed into it; always 0

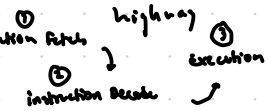
use cases:
- clear or ignore data

Examples
```
ADD R0, R1, R2
CMP R1, R2      ] use R0 as condition
BEQ label
LDR R0, [R3]    ] load data
```

Datapath?

* hardware components that perform operations like arithmetic, logic, and data movement
* collection of functional units & interconnections that perform data processing and on CPU

ALU, Registers, Multiplexers, Memory Access Units, Buses, Program Counter, Instruction Register



"hardware" of CPU

Types of Datapaths:

- Single-cycle:
  - Executes entire instruction in one cycle
  - simple, but inefficient for complex instructions
- multi-cycle:
  - Breaks instructions into smaller tasks, each taking one clock cycle
  - complex, but more efficient
- *. pipelined:
  - common in modern CPUs
  - overlaps multiple instructions to improve throughput

Project: Comp. Arch.

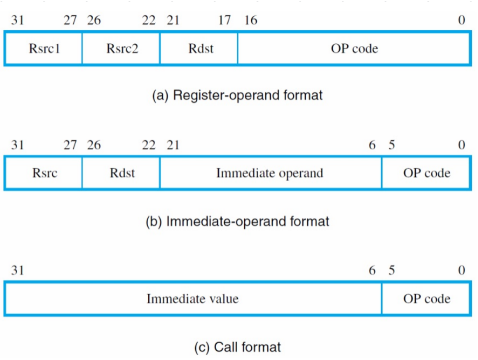
1. Create top level SISC.v file which instantiates & connects various components of datapath.
2. Design control unit that will provide various control signals to the other components

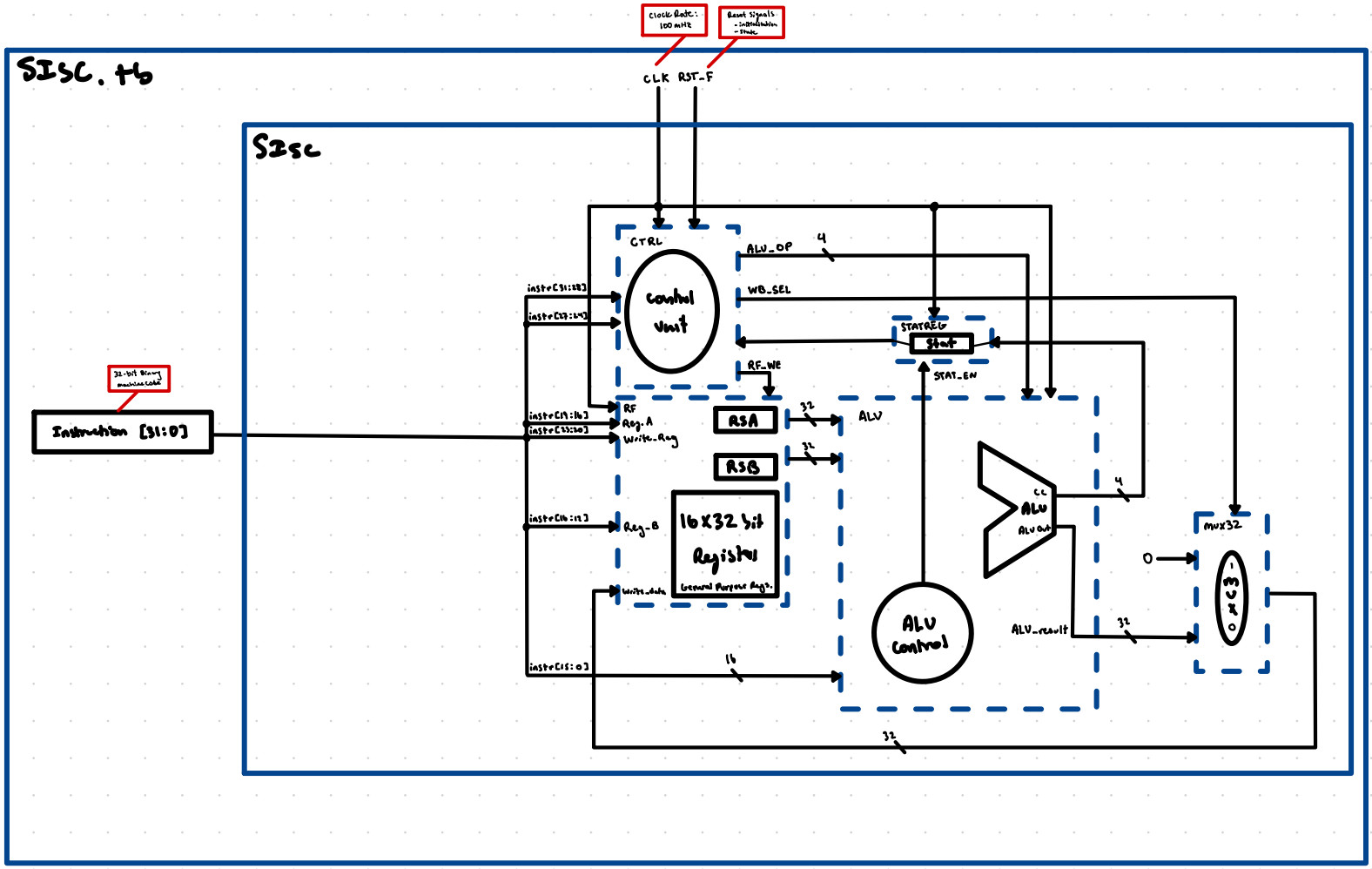
Phase 1:

- Implement Datapath
- verify correct operation

TODO:

1. Implement register & immediate operand instructions
2. create testbench file to instantiate SISC.v file
3. Provide testbench with reset and clock signals
4. Provide set of simulated instructions to datapath





ctrl.v code

① pos clk / reset check

if rst_f ≠ 0 → move states

② combinational procedure

...

case (present_state)

actual start

Start 0 (initial) → Start 1

Start 1 → fetch

fetch → decode

decode → execute

execute → mem

mem → writeback

Parameters

. constants of present state

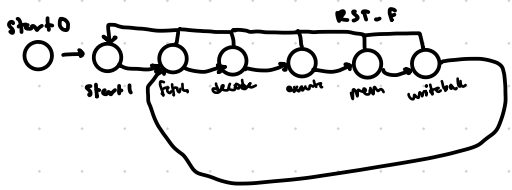
start 0 : 0

start 1 : 1

fetch : 2

...

writeback : 6



③ when present state or opcode changes

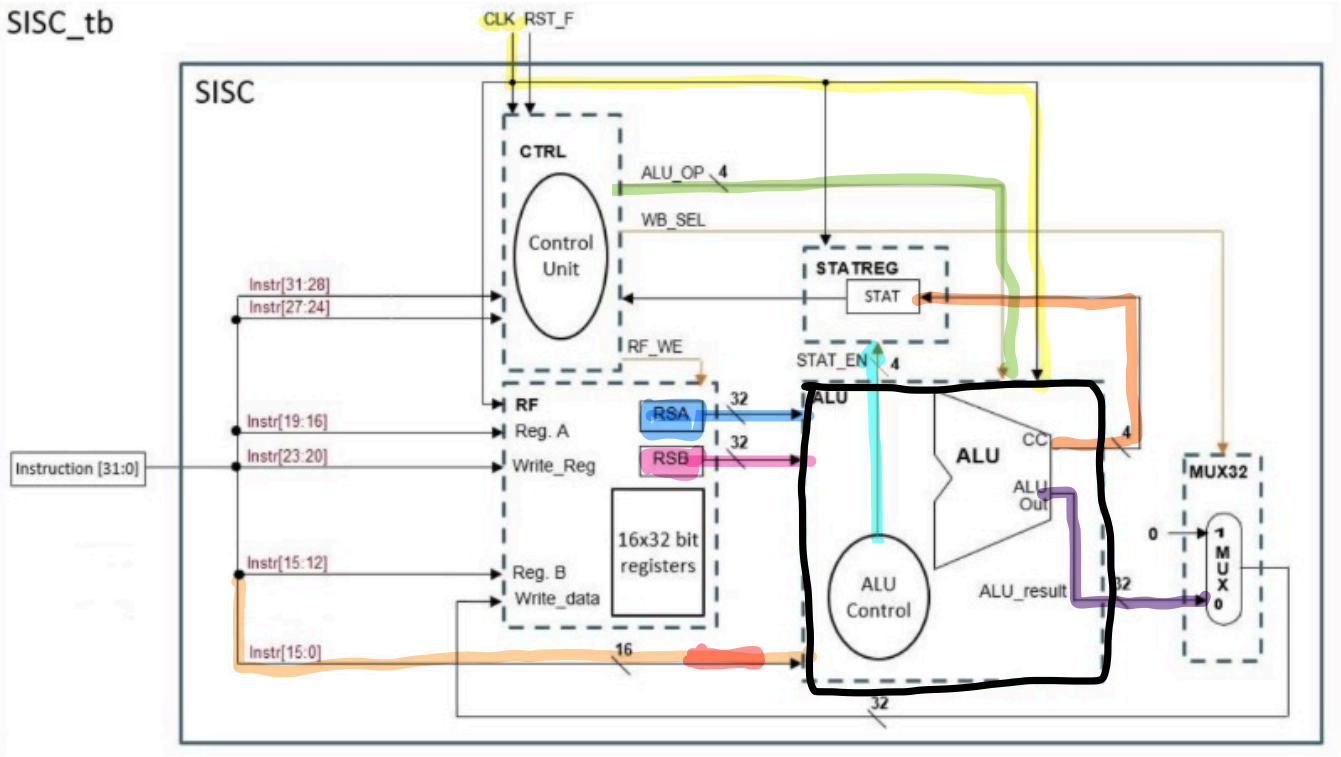
default values:

rf_we : 0

wb_sel : 0 default write back

alu_op : NOP 0000

Component: ALU



inputs:

clk : system clock

rsa : operand A via register file (32-bits)

rsb : operand B via register file (32-bits)

imm: immediate value to be sign extended (16-bits) → 32-bits inside ALU

alu_op : controls operation to be performed by ALU (4 bits)

Bit 0: 1: update status reg
0: don't update status reg

Bits [3:1] : specify operation

funct: controls function to be performed (4 bits)

outputs:

alu_result : output from ALU (32-bits) & latched on positive edge of clock

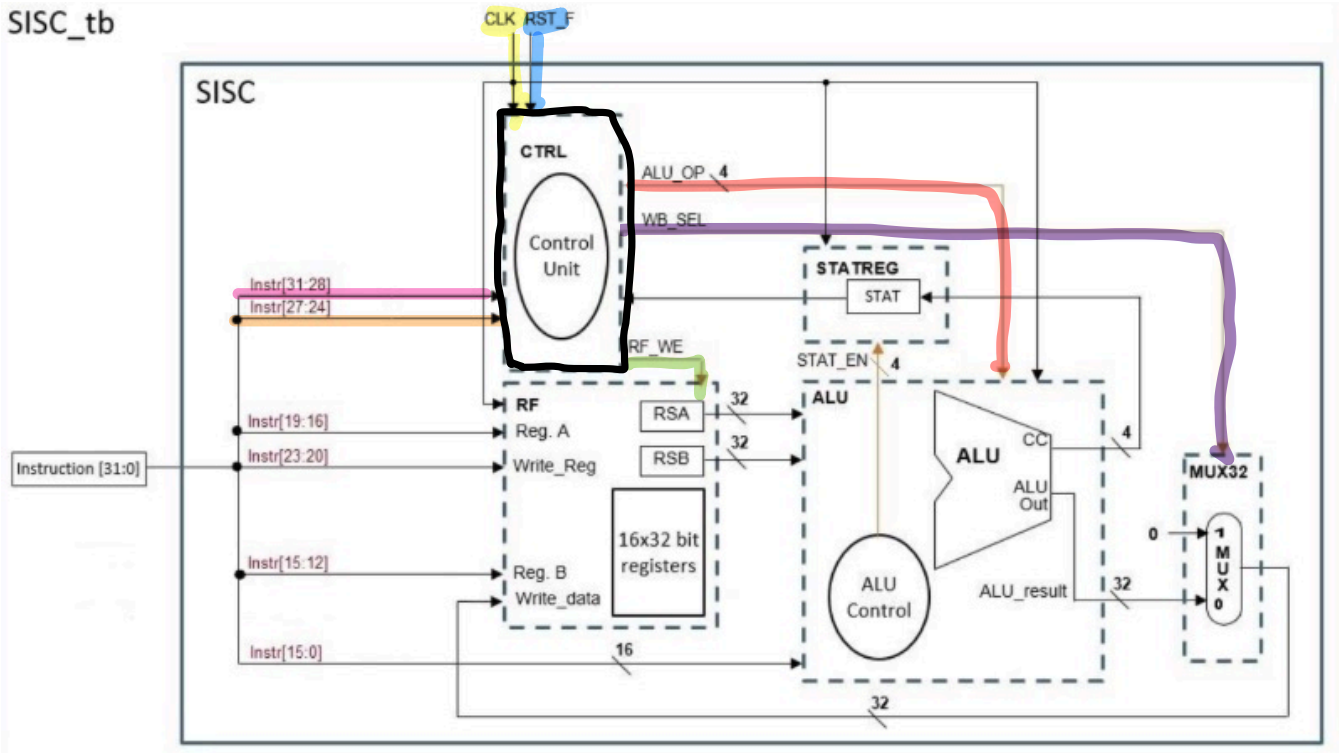
stat : status flags (4-bit)

B3 : carry [C]
B2: overflow [V]
B1 : negative [N]
B0: zero [Z]

stat_en : controls if status bits from output should be saved (4-bits)

B3 : carry [C]
B2: overflow [V]
B1 : negative [N]
B0: zero [Z]

Component: control unit



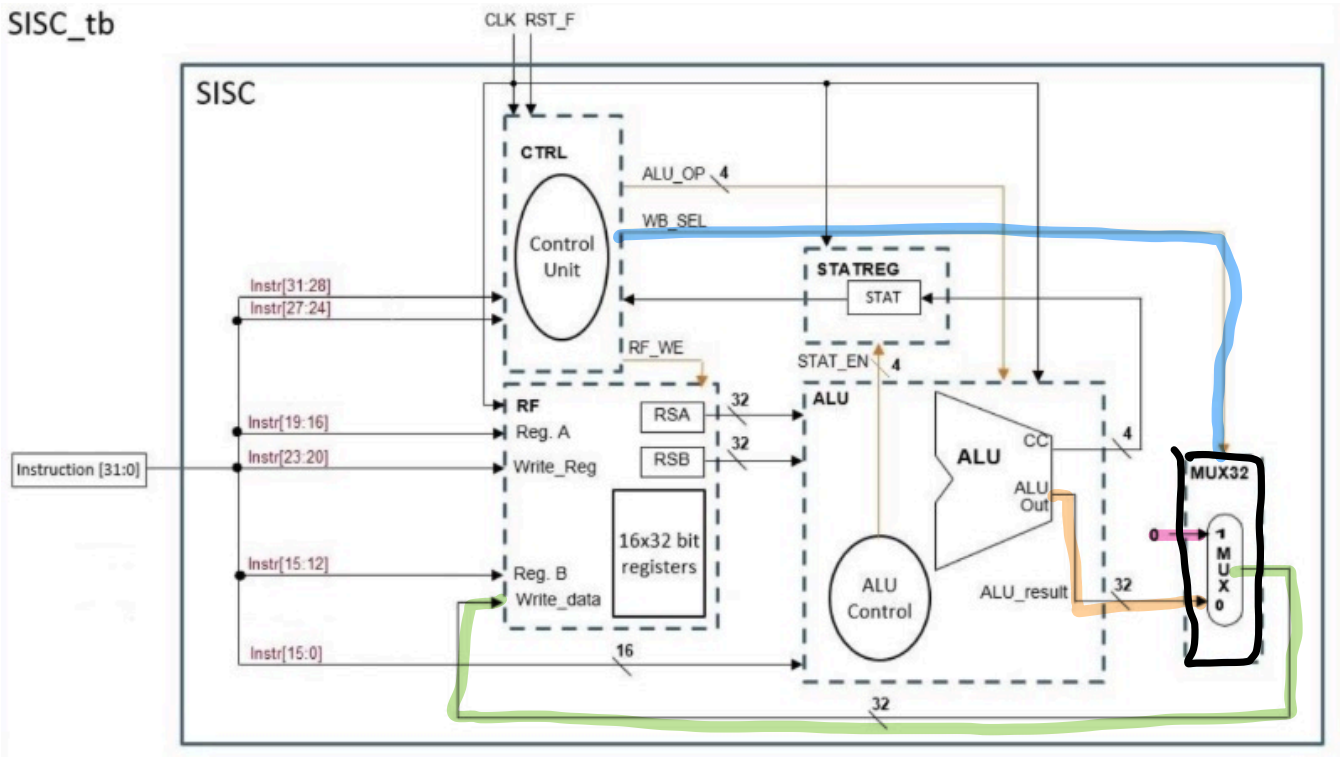
Inputs:

- clk: system clock 1 bit
- RST_F: reset signal 1 bit
- opcode: operation to be performed 4 bit
- mm: memory mode 4 bit

outputs:

- rf_we: register file write enable
- alu_op: alu opcode
- wb_sel: writeback select for source register

Component: mux32



inputs:

wb_sel: select which input propagates to my writeback (1 bit)

0: no work

1: yes write back

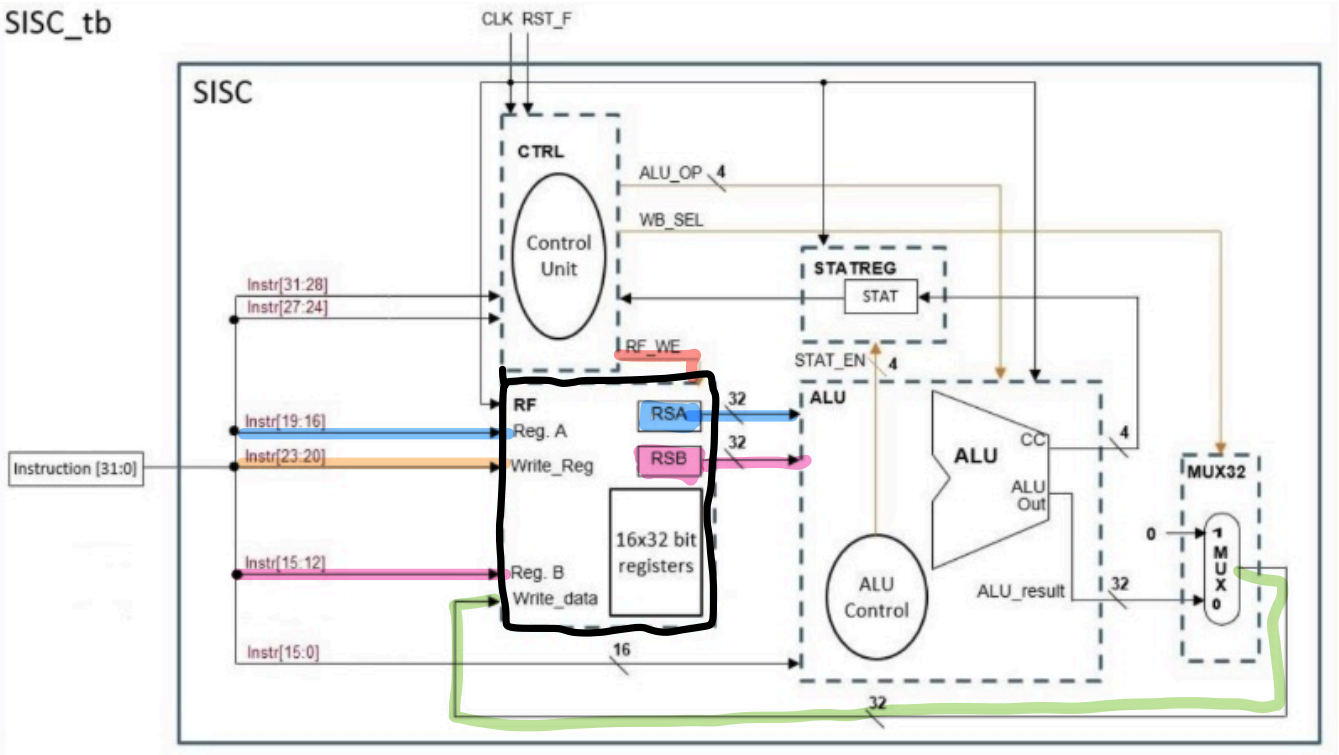
in_A: first input default 0x0000 0000 0000 0000 (32 bits)

in_B: second input Output from ALU result (32 bits)

outputs:

out: multiplexor output to be written back to register

component: Register file



inputs:

read_rega : address for reg A value (4 bits)

read_regb : address for reg B value (4 bits)

write_reg : address of register to write to (4 bits)

write_data : data to write to register write_reg address (32 bits)
rf_we must be 1

rf_we : enable writeback

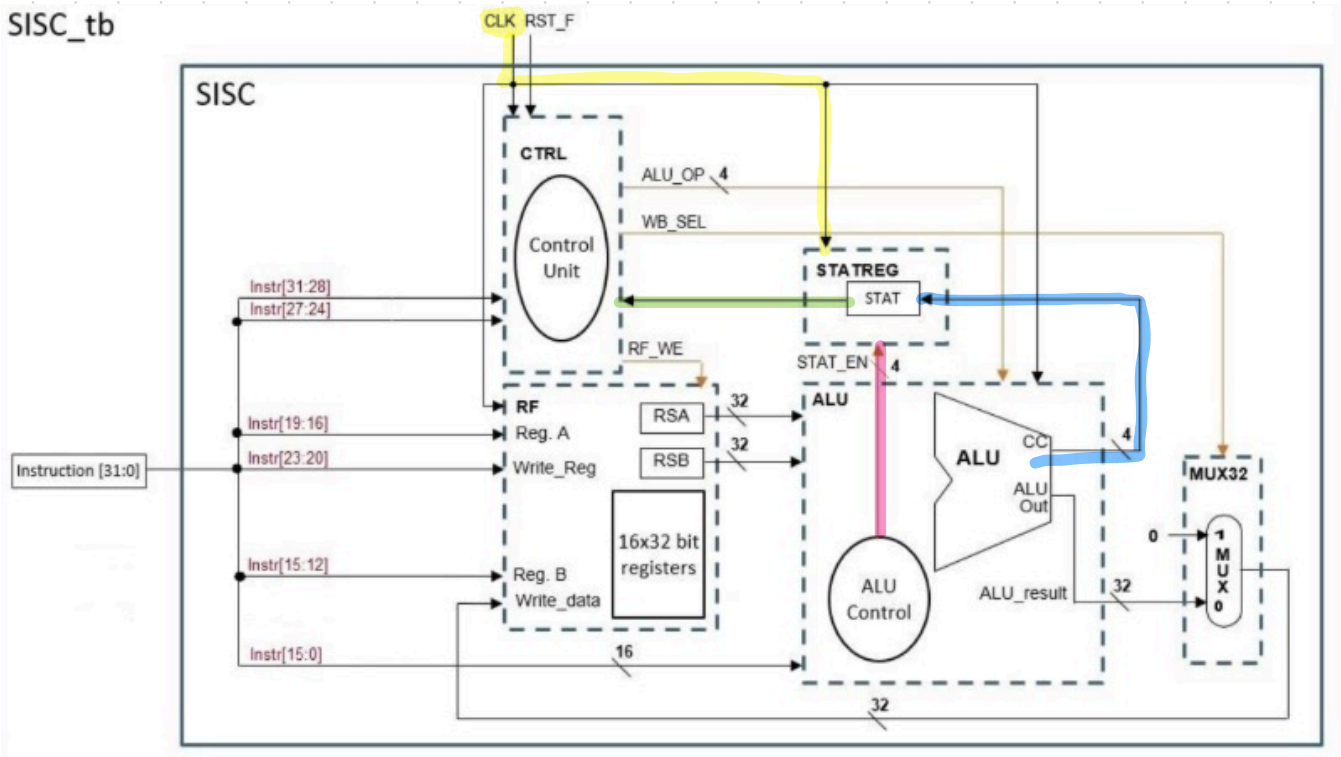
outputs:

rsa : register a into ALU contents of read_rega (32 bit)

rsb : register b into ALU contents of read_regb (32 bit)

note these are latched on positive edge of clock

Component: Status Register



inputs:

clk: system clock (1 bit) (pos edge)

in: status register bits from ALU operation (4 bits)

enable: for each bit in in bits update status register CC Flags (4 bits)

outputs:

out: status bits saved from ALU (4 bits)