EE 224: RISC CPU

Mihir Kavishwar (17D070004)
Rishabh Dahale (17D070008)
Mithilesh Vaidya (17D070011)
Anubhav Agarwal (17D070026)

April 2019

1 Overview

We have designed a simple CPU which can implement 14 basic general purpose instructions. These consists of addition, bit-wise NAND, control flow and reading, modifying and navigating through memory. The only input to the ALU is the clock and a reset input which initialises the required signals to it's starting value. The design of the desired general purpose computing system consists of the following components:

1. Read and Write Memory
2. Register File (RF)
3. Arithmetic Logic Unit (ALU)
4. Control Unit

We have implemented a finite state machine in the control unit which is responsible for the state transitions, output signals and controlling the various sub components.

2 Arithmetic Logic Unit

The ALU has two 16 bit inputs 'A' and 'B' and a two bit control input which determines what operation to perform. In addition to the computed 16-bit value, it also outputs three 1-bit signals:

- C flag - set to 1 if addition results in overflow.
- Z flag - set to 1 if the 16-bit output is zero.
- X flag - set to 1 when ALU output is 8. It is used for the Load All/Store All instruction which has to be executed 8 times - once for each register in the register file.

ALU performs the following functions:

1. ADD - Addition of two 16-bit signed numbers with a carry out.
2. XOR - Bit-wise XOR of two 16-bit numbers. This part is used to check if two input numbers to the ALU are equal (Useful in BEQ instruction to detect if the two inputs are the same).
3. Bitwise NAND operation

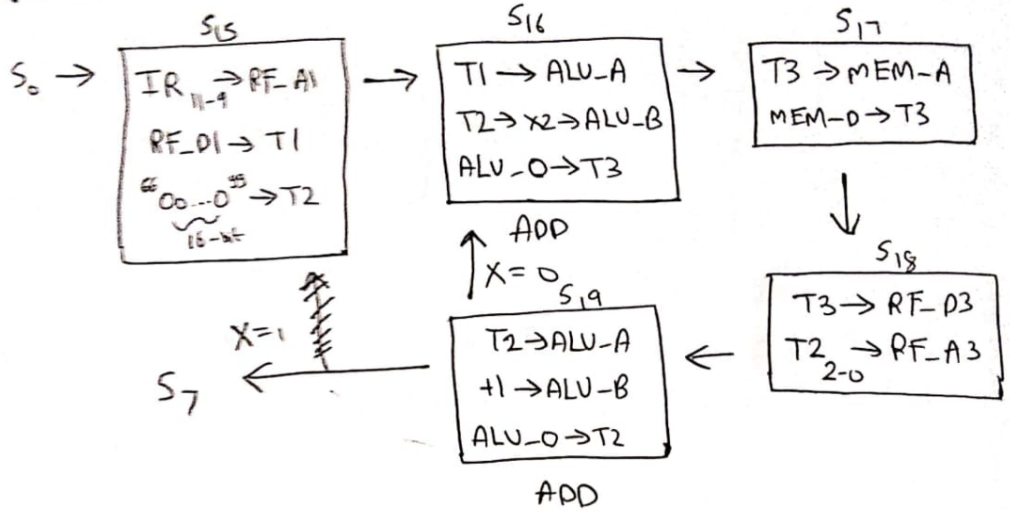
3 Register File

The register file consists of eight 16-bit registers. The RF stores the data provided at the port RF_D3 into location provided at port RF_A3, at every rising edge of clock cycle.
The read operation has been implemented as a combinational logic and write operation is implemented as behavioral logic using process statements.

4 Control Unit(CPU)

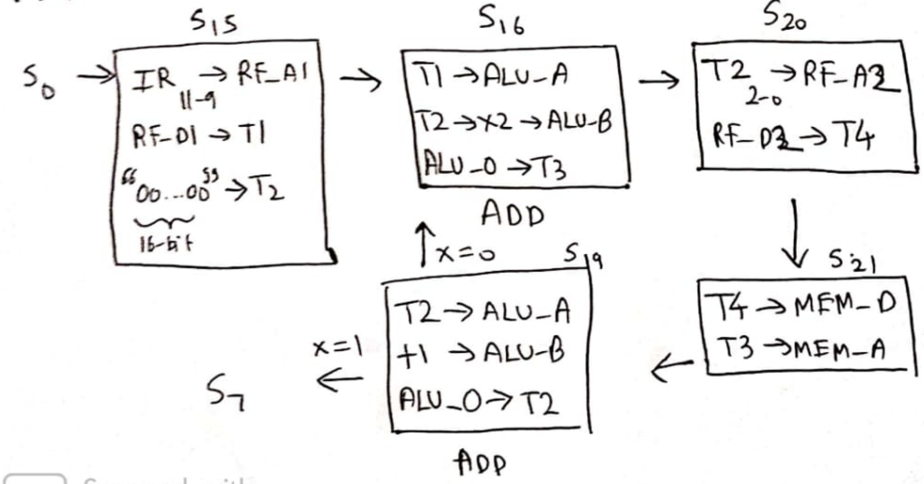
This is the heart of our machine. It integrates the various sub-components (Memory, ALU and Register File). It also has temporary registers and MUXes that are used to combine the various sub-components into a single datapath.
We have implemented the finite state machine in this part using behavioural VHDL logic. The FSM has various control and enable signals for the sub-components as its output. It takes the OpCode and values of C,Z registers as input in its output logic.
Our machine is of Mealy type with 30 different states. The machine is Mealy since the output(i.e control signals) depends on the input(i.e OpCode and C,Z) The operations that take place in each state are shown in the figures below. The specification of each instruction can be found here.

* LA:



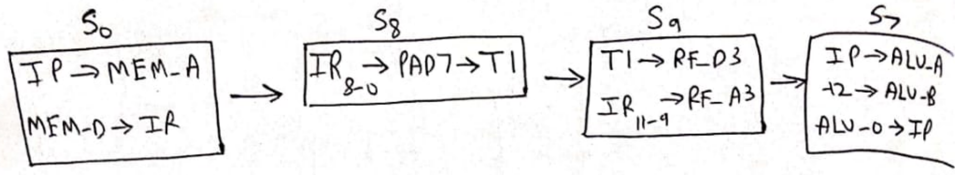
S_0
S_{15}
$IR_{11-9} \to RF\_A1$
$RF\_D1 \to T1$
"00...0" → T2
16-bit
S_{16}
T1 → ALU_A
T2 → x2 → ALU_B
ALU_O → T3
ADD
S_{17}
T3 → MEM_A
MEM_D → T3
S_{18}
T3 → RF_D3
$T2_{2-0} \to RF\_A3$
S_{19}
T2 → ALU_A
+1 → ALU_B
ALU_O → T2
ADD
X=0
X=1
S_7

* SA:

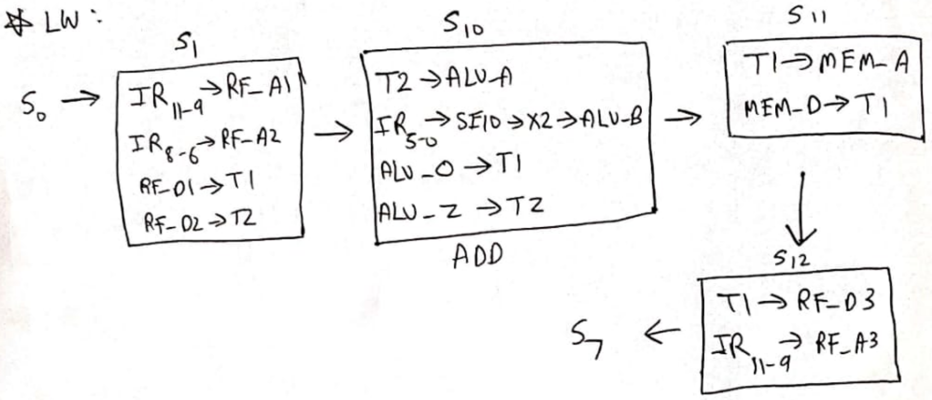


S_0
S_{15}
$IR_{11-9} \to RF\_A1$
RF_D1 → T1
"00...00" → T_2
16-bit
S_{16}
T1 → ALU_A
T2 → x2 → ALU_B
ALU_O → T3
ADD
S_{20}
$T2_{2-0} \to RF\_A2$
RF_D2 → T4
S_{21}
T4 → MEM_D
T3 → MEM_A
S_{19}
T2 → ALU_A
+1 → ALU_B
ALU_O → T2
ADD
x=0
x=1
S_7

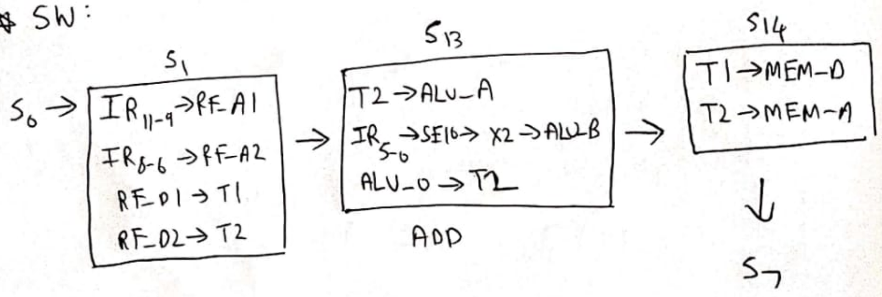
☆ LHI :


S0
IP → MEM_A
MEM_D → IR
S8
IR8-0 → PAD7 → T1
S9
T1 → RF_D3
IR11-9 → RF_A3
S7
IP → ALU_A
+2 → ALU_B
ALU_O → IP

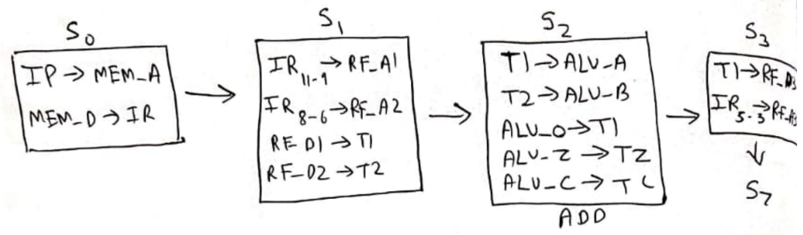
☆ LW :


S0
S1
IR11-9 → RF_A1
IR8-6 → RF_A2
RF_D1 → T1
RF_D2 → T2
S10
T2 → ALU_A
IR5-0 → SE10 → X2 → ALU_B
ALU_O → T1
ALU_Z → T2
ADD
S11
T1 → MEM_A
MEM_D → T1
S12
T1 → RF_D3
IR11-9 → RF_A3
S7

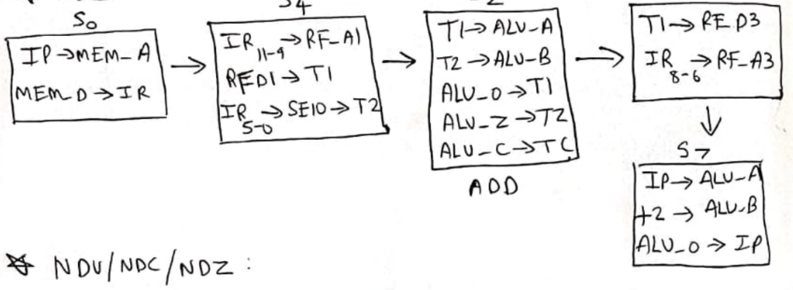
☆ SW :


S0
S1
IR11-9 → RF_A1
IR8-6 → RF_A2
RF_D1 → T1
RF_D2 → T2
S13
T2 → ALU_A
IR5-0 → SE10 → X2 → ALU_B
ALU_O → T2
ADD
S14
T1 → MEM_D
T2 → MEM_A
S7

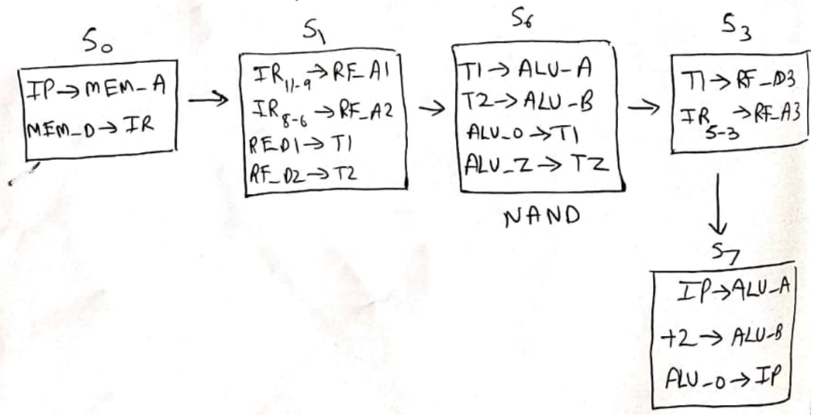
☆ ADD/ADC/ADZ :



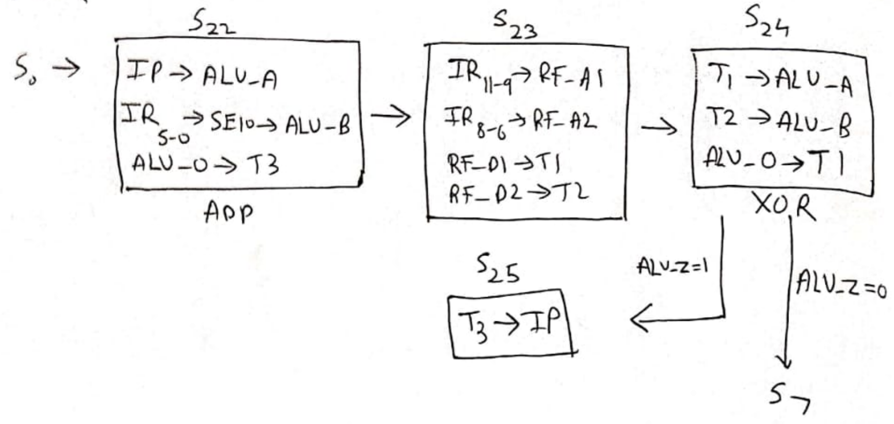
☆ ADI :



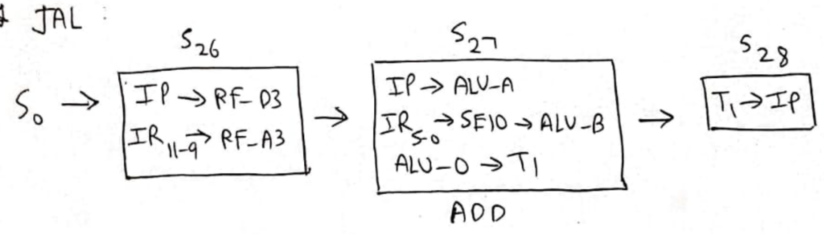
☆ NDU/NDC/NDZ :



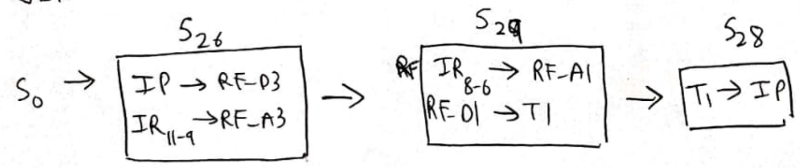
✯ BEQ :


S_0
S_{22}
$IP \to ALU\_A$
$IR_{5-0} \to SE10 \to ALU\_B$
$ALU\_O \to T3$
ADD
S_{23}
$IR_{11-9} \to RF\_A1$
$IR_{8-6} \to RF\_A2$
$RF\_D1 \to T1$
$RF\_D2 \to T2$
S_{24}
$T_1 \to ALU\_A$
$T2 \to ALU\_B$
$ALU\_O \to T1$
XOR
ALU_Z=1
ALU_Z=0
S_{25}
$T_3 \to IP$
S_7

✯ JAL :


S_0
S_{26}
$IP \to RF\_D3$
$IR_{11-9} \to RF\_A3$
S_{27}
$IP \to ALU\_A$
$IR_{5-0} \to SE10 \to ALU\_B$
$ALU\_O \to T1$
ADD
S_{28}
$T_1 \to IP$

✯ JLR :


S_0
S_{26}
$IP \to RF\_D3$
$IR_{11-9} \to RF\_A3$
S_{29}
~~RF~~ $IR_{8-6} \to RF\_A1$
$RF\_D1 \to T1$
S_{28}
$T_1 \to IP$

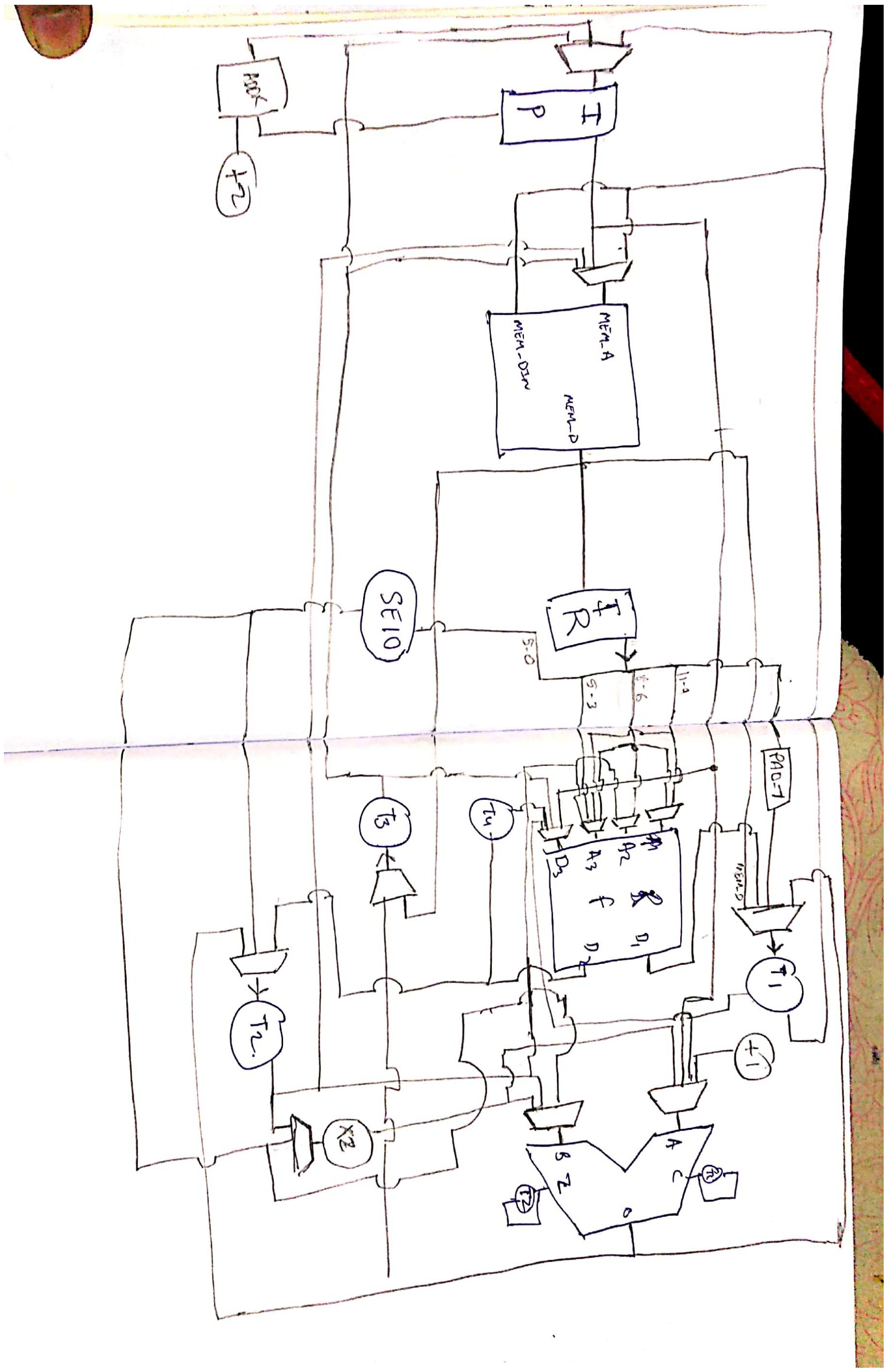


Figure 1: The Datapath (Excuse the clumsiness!)

Note:

- The x2 operation in states S_{10} and S_{16} is unnecessary. (It is required when memory is byte-addressable as opposed to the 16-bit addressable-memory which has been implemented)
- 'PAD7' refers to the operation of placing the given 9 bits as MSB and padding "0000000" to make it a 16-bit vector.
- SE10 and SE7 are sign extenders - They are used to convert a 6-bit or a 9-bit vector into a 16-bit vector without altering it's value. It is padded with 1s or 0s depending on the first bit of the input.

The CPU consists of the following components:

1. ALU
2. Register File
3. Memory Unit
4. 10 bit sign extender
5. 7 bit sign extender

The CPU has signals to store contents of instruction register, instruction pointer and 5 temporary registers. Signals are also declared to map components like ALU to the CPU.

5 Testing

To test our machine, we stored the instruction in memory, initialised the Instruction Pointer to 0 and checked the memory and register file after each instruction to ensure that it is working as desired.