CSN 221: Computer Architecture and Microprocessor

PROJECT 3: SimpleRISC PROCESSOR

<u>REPORT FILE</u>

GROUP MEMBERS

- Aniket Umesh Kathare (20114010)
- Arnav Vyas (20114015)
- Deepak Tailor (20114029)
- Deepesh Garg (20114032)
- Dighiya Sanidhya Ramjiprasad (20114035)
- Jyoti Chauhan (20114042)
- Tippana Rajesh (20114101)
- Doraiswamy R Harshavardhan (18114022)

THE PROBLEM STATEMENT

Design the SimpleRISC processor discussed in the class on Logisim simulator. Extend the basic ideas discussed in the class with several other ideas that you seem important to be included. Preferably, to enable multicore environment. Evaluate rigorously. BENCHMARK evaluations are preferred.

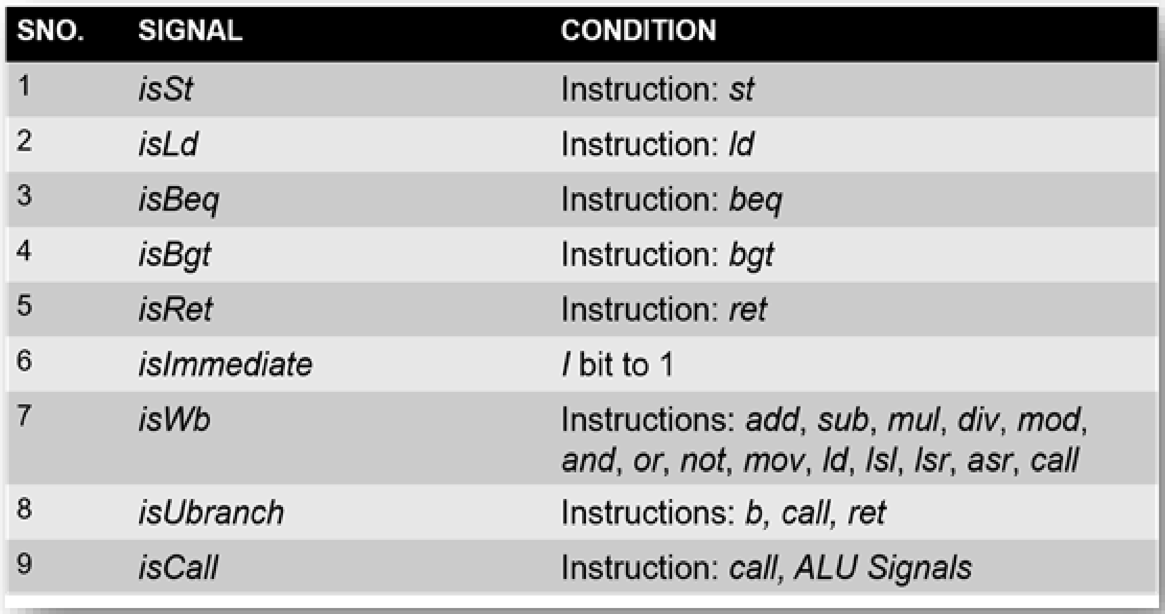
INDIVIDUAL CONTRIBUTIONS

- Aniket Umesh Kathare: Main processor & Control Unit, Report preparation
- Arnav Vyas: Main Processor & Control Unit, Report preparation
- Deepesh Garg: Register Files & Flags Unit, Report preparation
- Deepak Tailor: Register Files & Flags Unit, Report preparation
- Dighiya Sanidhya Ramjiprasad: Arithmetic Logical Unit & Immediate value calculation, PPT file
- Jyoti Chauhan: Arithmetic Logical Unit & Immediate value calculation, PPT file
- Tippana Rajesh: Branch Unit & Memory Unit, Report preparation
- Doraiswamy R Harshavardhan: Simple RISC processor & ISA design method

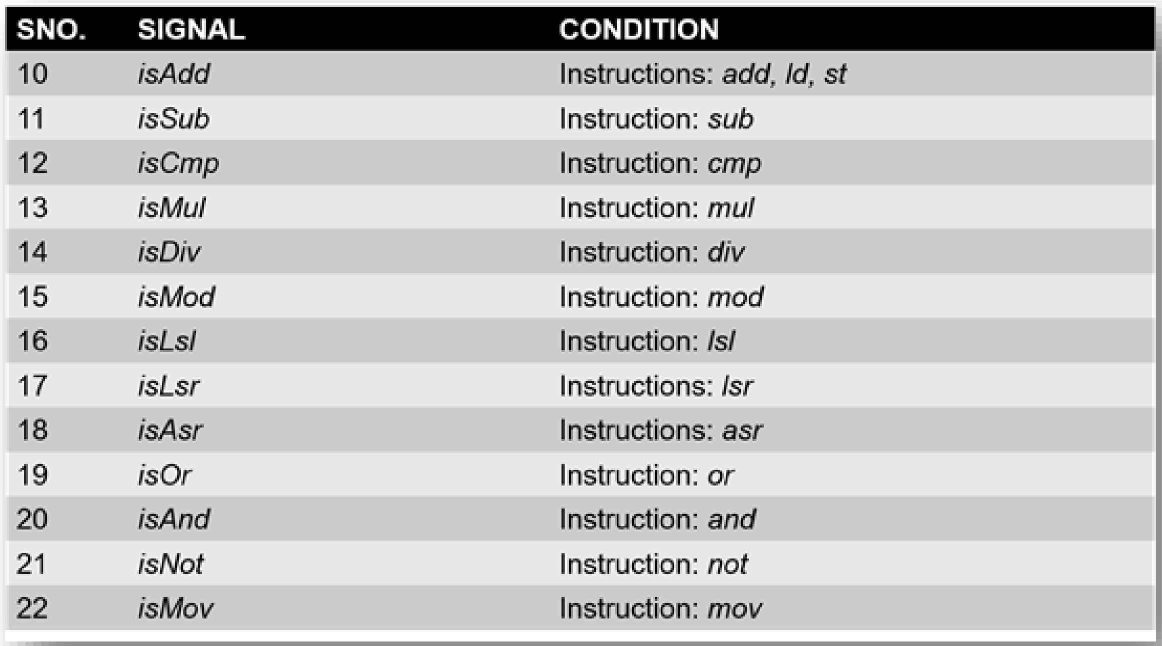
NOVELTY OF THE WORK DONE

- All the SimpleRISC instructions which have been discussed in lectures are used.
- It contains 21 instructions.
- The complete processor is divided into 5 stages.
- The processor is made using Logisim.
- We can run any program using this processor having SimpleRISC instructions to give us the correct outputs.
- We have added a **display feature**, which displays the contents of any register.

Following are the instructions present in our ISA (Instruction Set Architecture).



SNO.	SIGNAL	CONDITION
1	*isSt*	Instruction: *st*
2	*isLd*	Instruction: *ld*
3	*isBeq*	Instruction: *beq*
4	*isBgt*	Instruction: *bgt*
5	*isRet*	Instruction: *ret*
6	*isImmediate*	*I* bit to 1
7	*isWb*	Instructions: *add*, *sub*, *mul*, *div*, *mod*, *and*, *or*, *not*, *mov*, *ld*, *lsl*, *lsr*, *asr*, *call*
8	*isUbranch*	Instructions: *b, call, ret*
9	*isCall*	Instruction: *call, ALU Signals*



SNO.	SIGNAL	CONDITION
10	*isAdd*	Instructions: *add, ld, st*
11	*isSub*	Instruction: *sub*
12	*isCmp*	Instruction: *cmp*
13	*isMul*	Instruction: *mul*
14	*isDiv*	Instruction: *div*
15	*isMod*	Instruction: *mod*
16	*isLsl*	Instruction: *lsl*
17	*isLsr*	Instructions: *lsr*
18	*isAsr*	Instructions: *asr*
19	*isOr*	Instruction: *or*
20	*isAnd*	Instruction: *and*
21	*isNot*	Instruction: *not*
22	*isMov*	Instruction: *mov*

METHODOLOGY & APPROACH

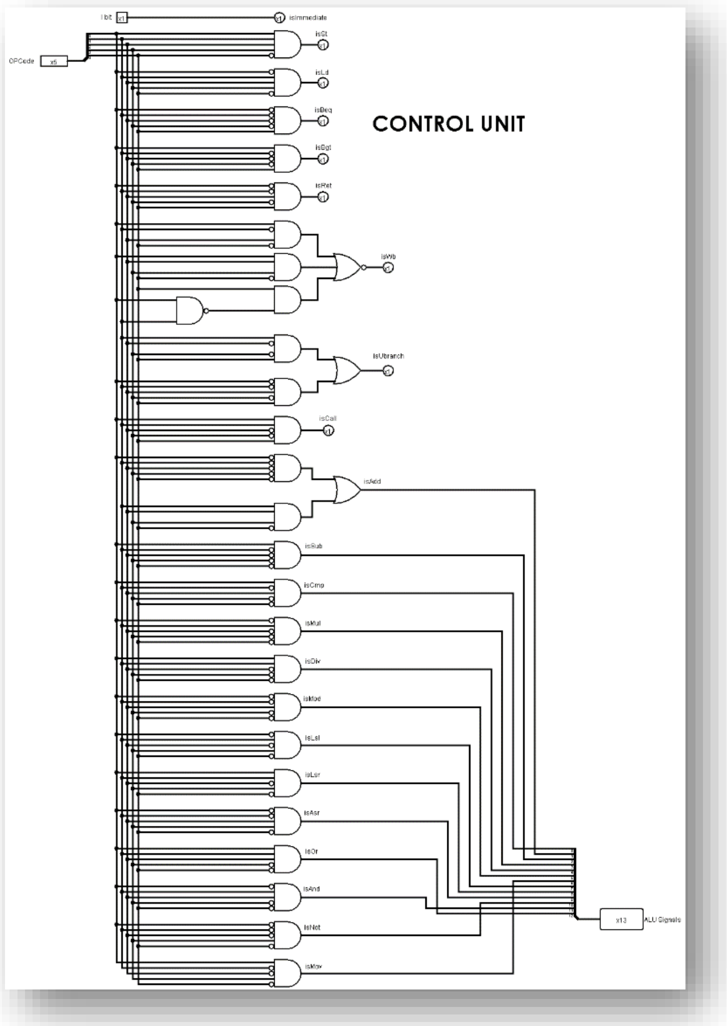
The aim of this project is to design a 32-bit SimpleRISC Processor. In doing so, we use a basic approach of dividing the processing into various stages and then design each stage.

We use multiplexers where alternate data sources are used for different instructions

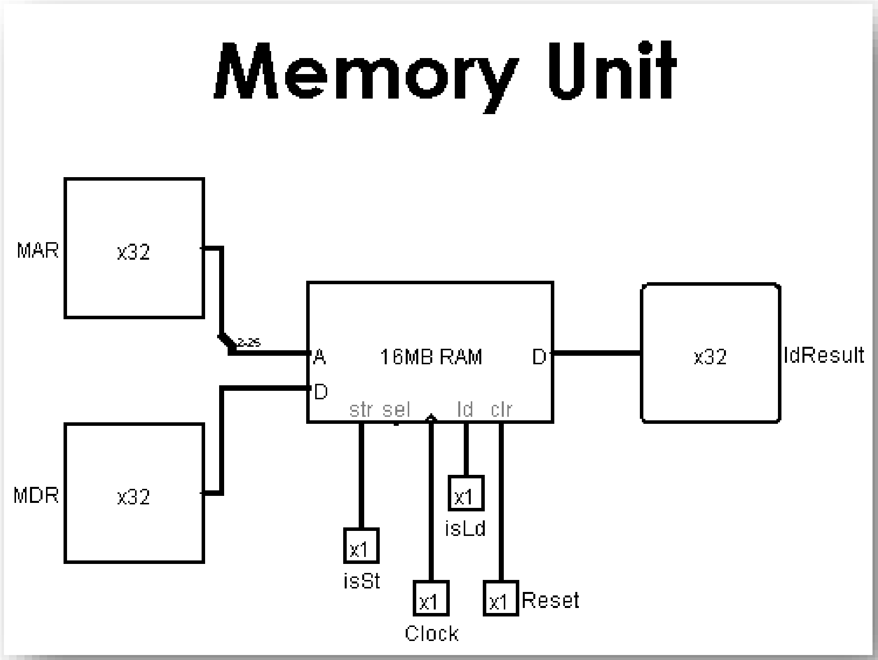
The instructions are executed systematically going through each stage. Every stage performs its pre-determined tasks and thus the whole task is completed separately.

The various stages are described as follows: -

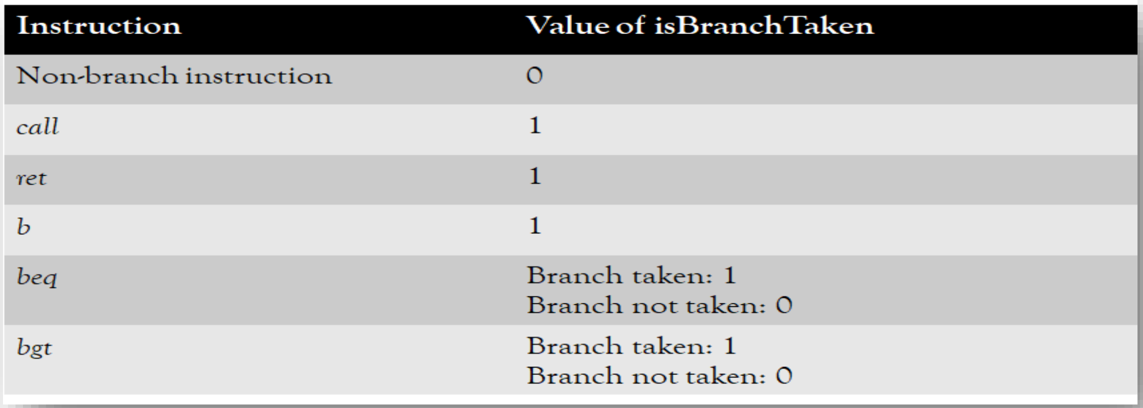
1. ***Control Unit*****-** This unit is given the opcode and the immediate bit. It generates all the control signals which regulate the working of processor's hardware.



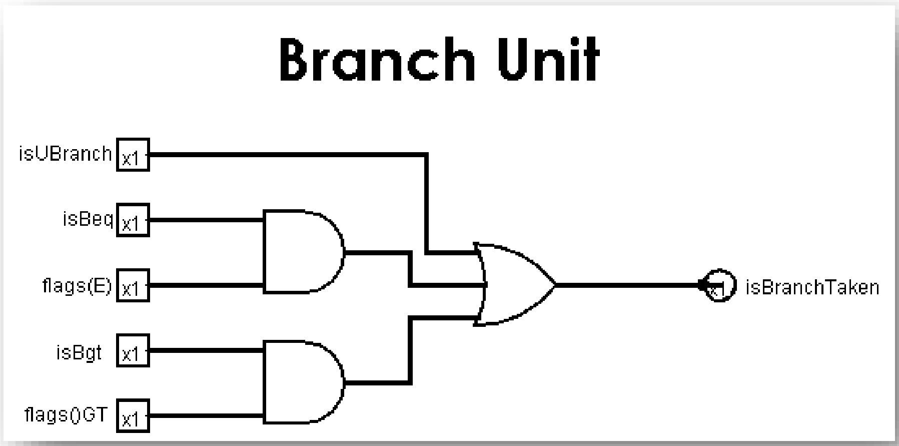
2. ***Memory Unit*-** It contains 2 registers mar and mdr. Mar buffers the memory address whereas mdr buffers the value that needs to be stored. §A set of arguments that specify the nature of operation such as load, or store is also required and finally data is in ldResult register after load operation is done.



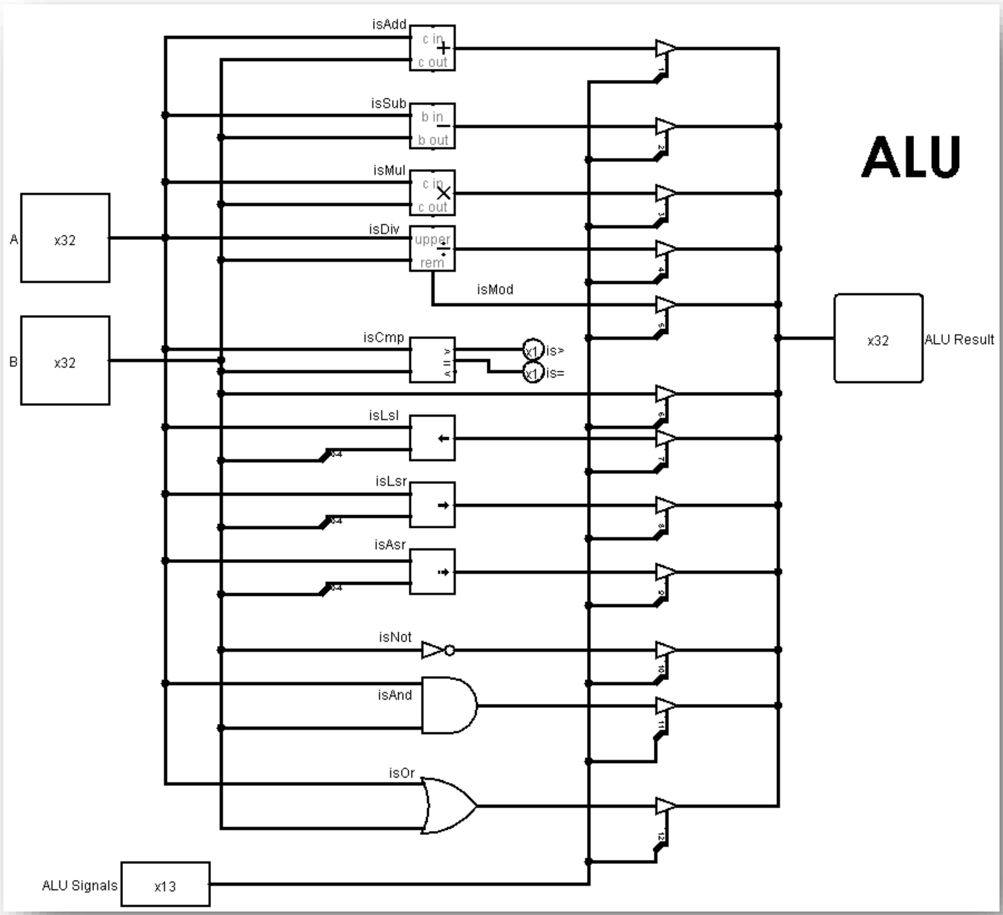
3. ***Branch Unit -*** This unit generates isBranchTaken signal which is a control signal for computing the branch condition (beq, bgt).



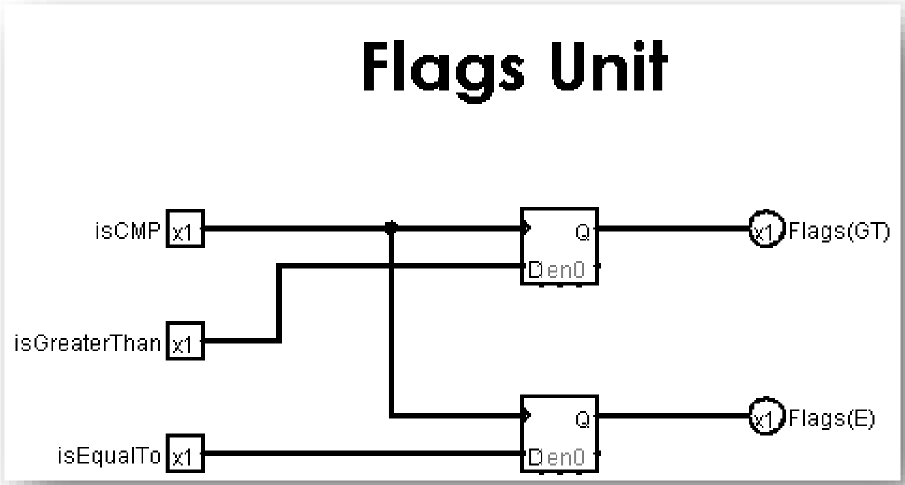
Instruction	Value of isBranchTaken
Non-branch instruction	0
call	1
ret	1
b	1
beq	Branch taken: 1 Branch not taken: 0
bgt	Branch taken: 1 Branch not taken: 0



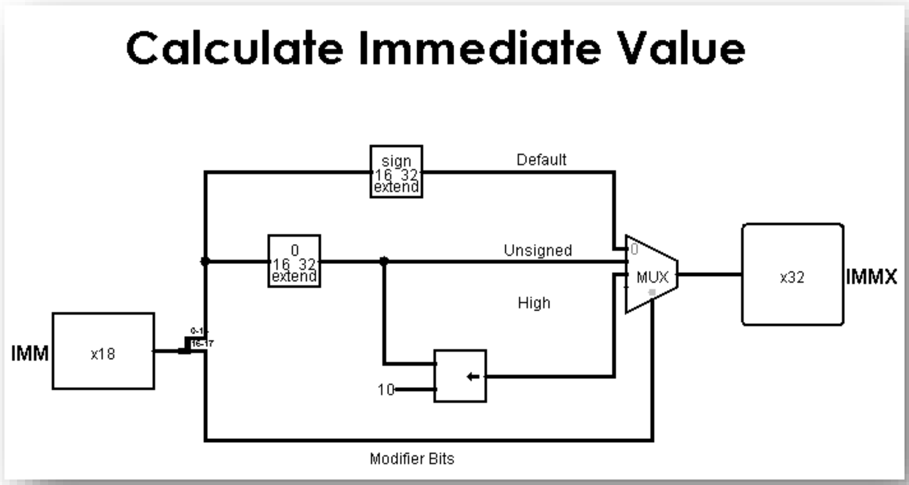
4. ***Arithmetic Logical Unit (ALU)***- It contains various modules, and we can enable or disable them as per requirement with the help of transmission gates. The various modules are: -
   a. Adder: isAdd, isSub, isCmp
   b. Multiplier: isMul
   c. Divider: isDiv, isMod
   d. Shift: isLsl, isLsr, isAsr
   e. Logical: isOr, isAnd, isNot



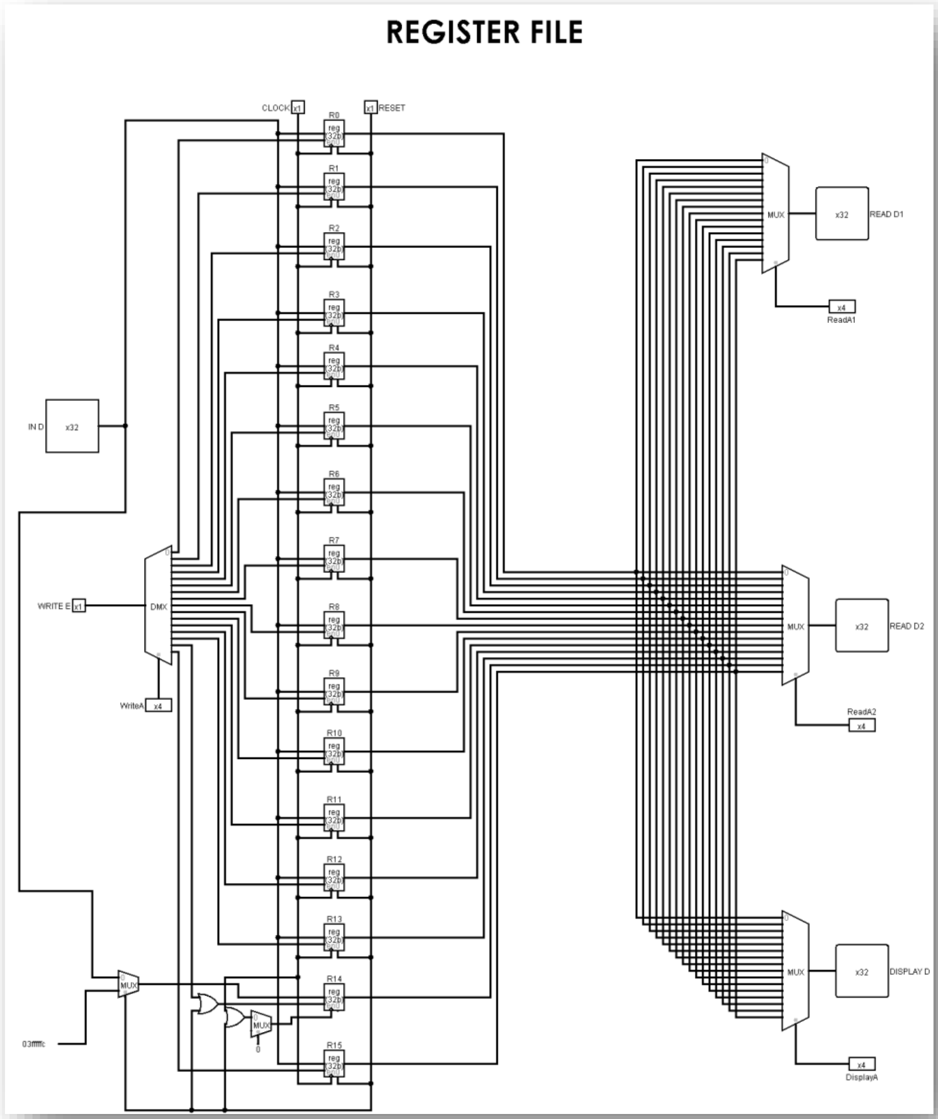
5. ***Flags Unit*** – The cmp instruction is a 2-address instruction that takes two source operands. The first source operand needs to be a register, and the second one can be an immediate or a register. It compares both the operands by subtracting the second from the first. It helps in execution of branch instruction.



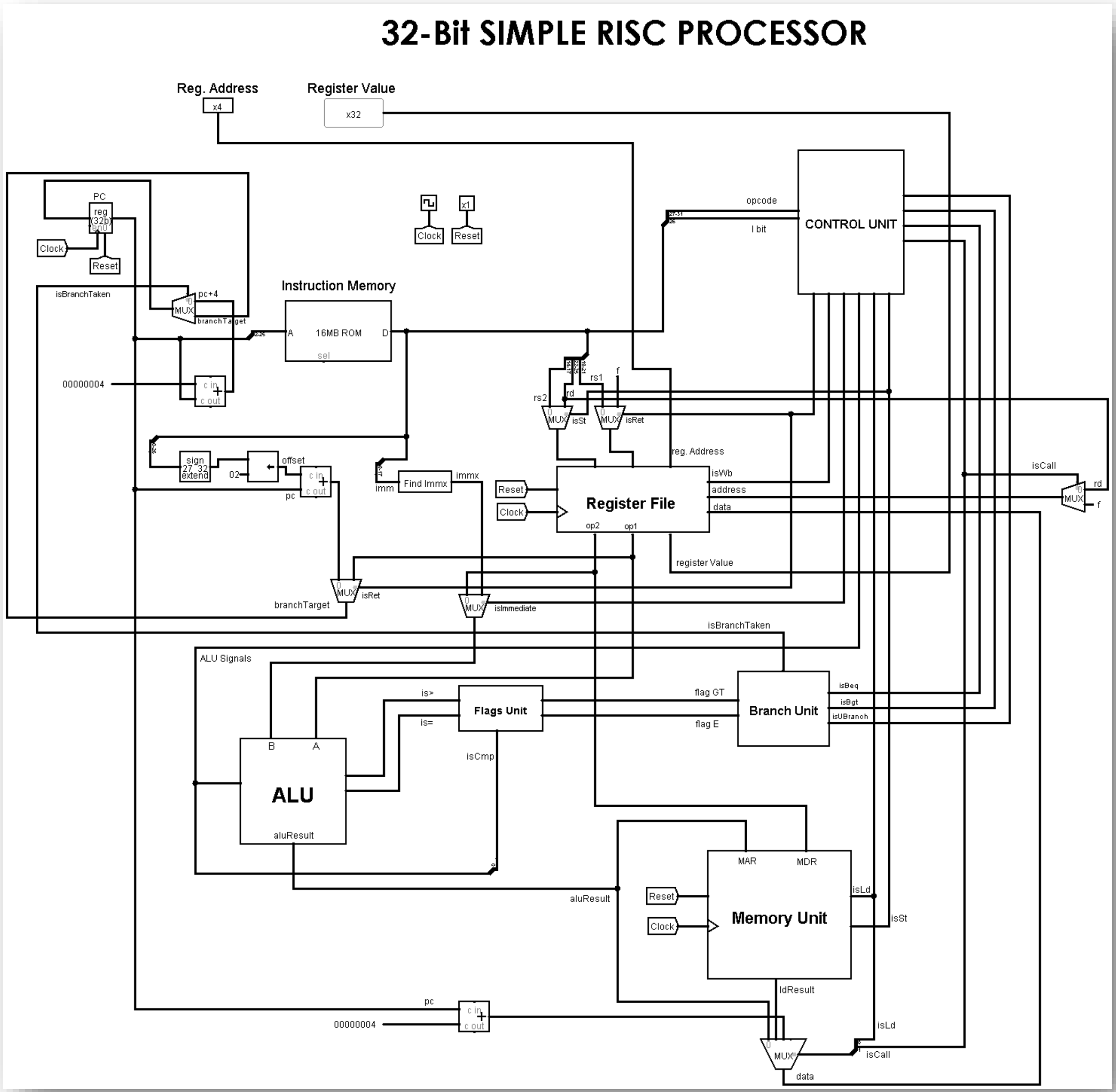
6. ***Immediate Value***- In computer architecture, a constant value specified in instruction is known as immediate. Out of 32 bits, we use 18 bits to specify the immediate (2 bits for modifier and 16 bits for constant part of immediate). The processor internally expands the immediate to a 32bit value, in accordance with the modifiers.



7. ***Register Files***-It contains 2 source registers: regSrc and regData. The regSrc register contains the number of the register that needs to be accessed whereas regData contains value to be written.
   - For write the value in regData is written to register specified by regSrc.
   - For read the register specified by regSrc is read and value is stored in ***regVal***.

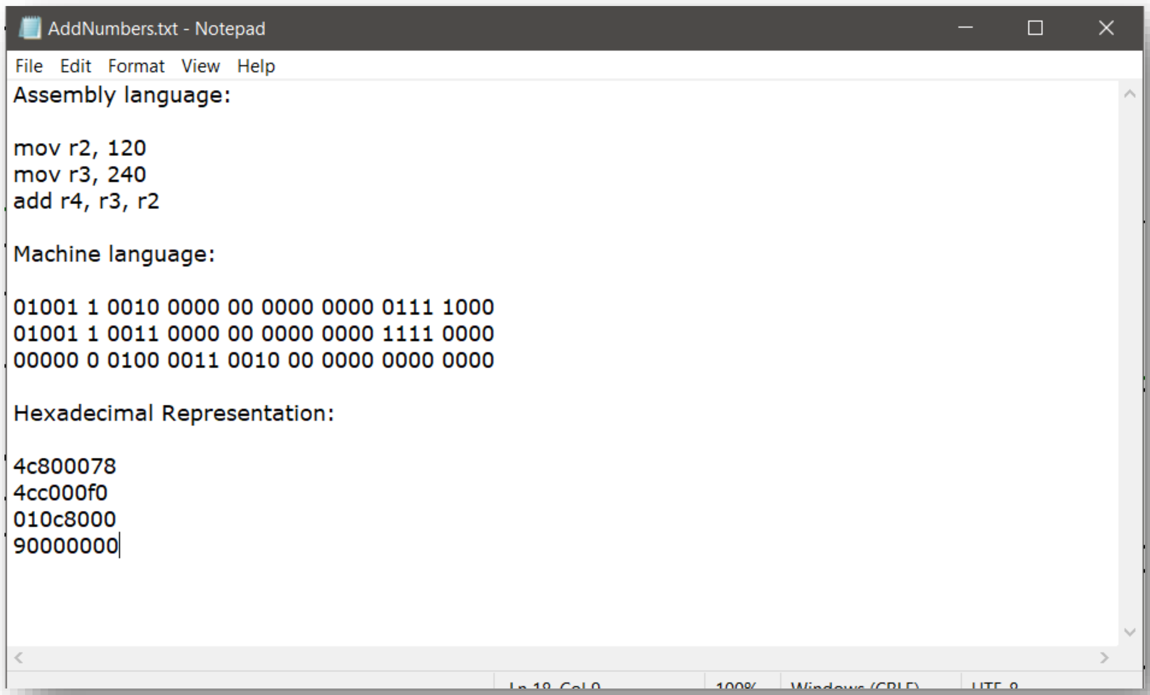


The final RISC processor is designed by connecting all the stages described above.



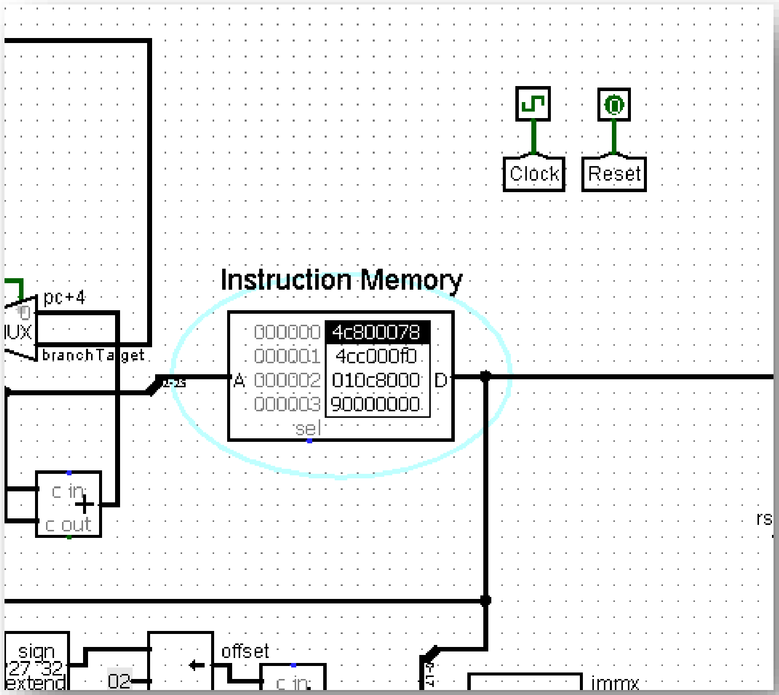
EVALUATION PARAMETER/TESTING

1. We can test a simple program as shown in snip below which adds the contents of two registers. The first five bits are representing the Opcode and the next bit is representing the Immediate bit. Other digits represent the register number (r0 register is represented as 0000, r1 as 0001, r2 as 0010 and so on) or Immediate value.

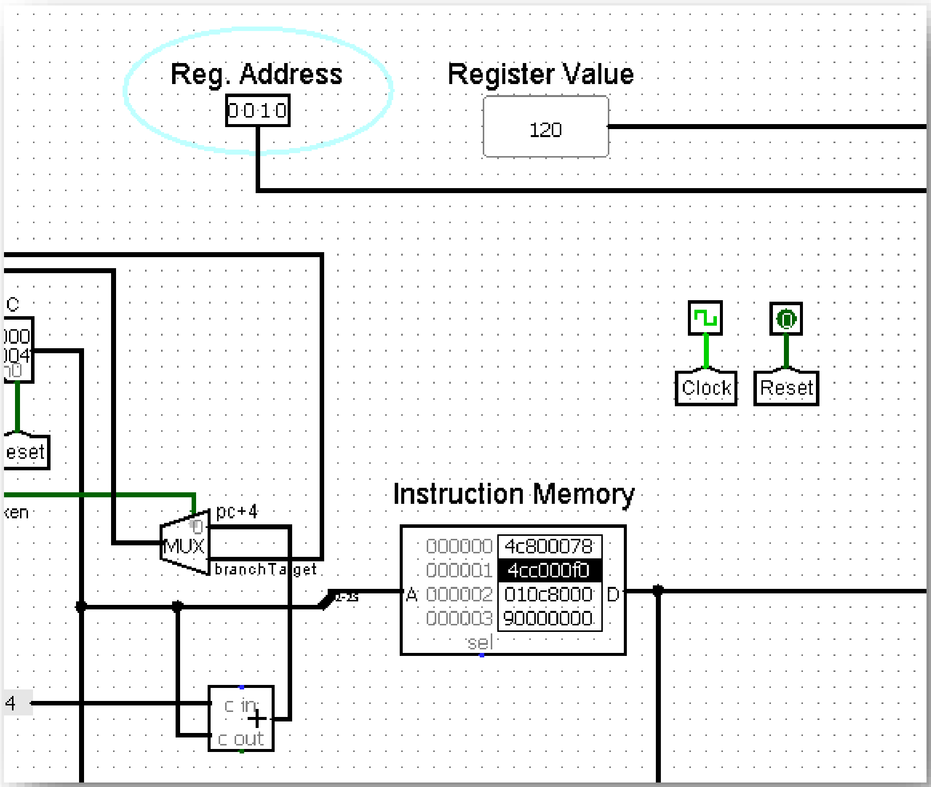


```
Assembly language:

mov r2, 120
mov r3, 240
add r4, r3, r2

Machine language:

01001 1 0010 0000 00 0000 0000 0111 1000
01001 1 0011 0000 00 0000 0000 1111 0000
00000 0 0100 0011 0010 00 0000 0000 0000

Hexadecimal Representation:

4c800078
4cc000f0
010c8000
90000000
```

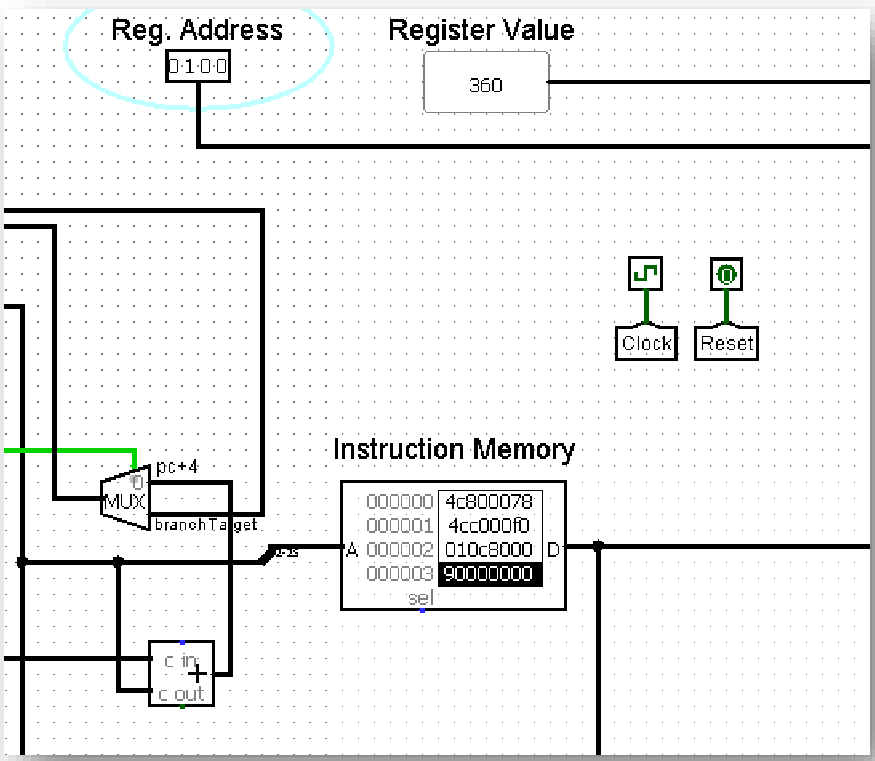
2. The Hexadecimal representation of the instructions is loaded in Instruction memory of our Main Processor. Here, 0x90000000 instruction tells the processor that program has ended. Below is the snip of hexadecimal instructions loaded in instruction memory.



3. Once the instructions are loaded, we need to activate the clock. Since it’s a single cycle processor it will carry out one instruction in a single clock cycle. Below is the snip of 0th instruction executed and the value of r2 getting updated as 120.



4. Once all the instructions are executed (in three cycles of clock), we can see in the below snip that value of r4 is gets updated as 360 (r4🡐r3+r2, 360 = 240 + 120).

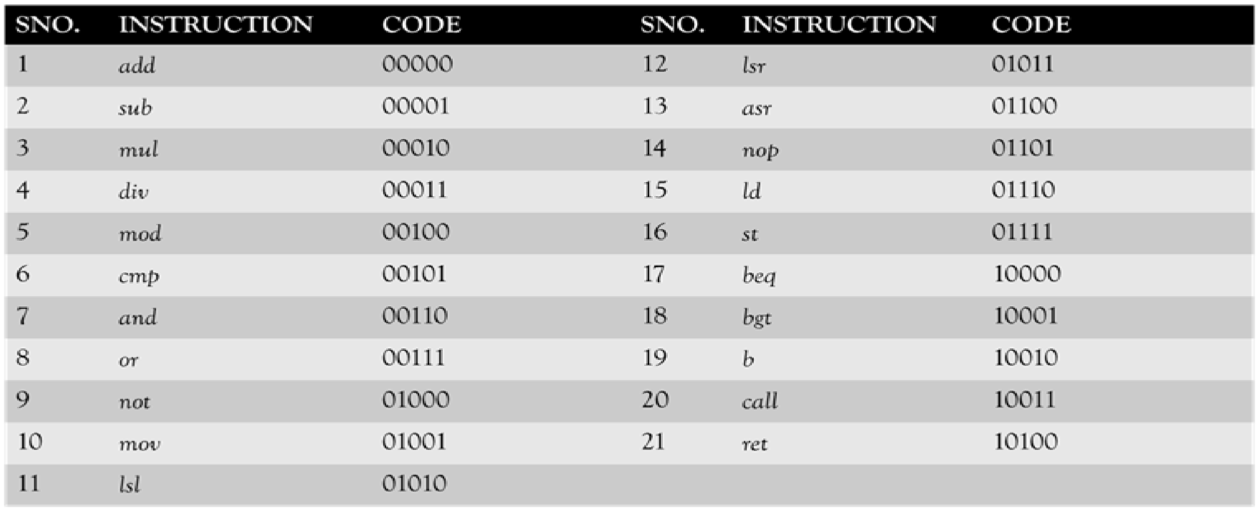


000000	4c800078
000001	4cc000f0
000002	010c8000
000003	90000000

RESULTS AND DISCUSSION

SimpleRISC processor is designed by Logisim.

The processor can run all the 21 microinstructions using opcodes as listed below:



SNO.	INSTRUCTION	CODE	SNO.	INSTRUCTION	CODE
1	*add*	00000	12	*lsr*	01011
2	*sub*	00001	13	*asr*	01100
3	*mul*	00010	14	*nop*	01101
4	*div*	00011	15	*ld*	01110
5	*mod*	00100	16	*st*	01111
6	*cmp*	00101	17	*beq*	10000
7	*and*	00110	18	*bgt*	10001
8	*or*	00111	19	*b*	10010
9	*not*	01000	20	*call*	10011
10	*mov*	01001	21	*ret*	10100
11	*lsl*	01010			

The processor contains different modules, and we can enable or disable them as per requirement with the help of transmission gates.

a. Adder: isAdd, isSub, isCmp
b. Multiplier: isMul
c. Divider: isDiv, isMod
d. Shift: isLsl, isLsr, isAsr
e. Logical: isOr, isAnd, isNot

CONCLUSION

A 32-bit SimpleRISC processor is designed using Logisim software.

RISC processor has ‘instruction sets’ that are simple and have simple ‘addressing modes. It is designed to perform a set of smaller computer instructions so that it can be operated at higher speeds (takes one clock per instruction).

A SimpleRISC ISA contains only 21 instructions.

RISC processor emphasizes on using the ***registers*** rather than memory because registers are the ‘fastest’ available memory source.

The processor is made by combining different processing units as following:

1. Main processor
2. Control unit
3. Set flags
4. Find immediate
5. ALU
6. Memory unit
7. Branch unit
8. Register file

REFERENCES

1. Computer Organization and Architecture

   Book by Dr. Smruti Ranjan Sarangi

2. RISC Processor

   https://binaryterms.com/risc-processor.html#RISCArchitecture

3. Processor Design: Part 1

   https://www.youtube.com/watch?v=EQGlk7UgcLs

4. SimpleRISC Processor:

   https://www.geeksforgeeks.org/computer-organization-and-architecture-pipelining-set-1-execution-stages-and-throughput/