Table of Contents

In this project, we designed a 32-bit pipelined CPU for the given SCU Instruction Set Architecture (SCU ISA). The SCU ISA is described below.

- Register file size: 64 registers, each register has 32 bits.
- PC: 32 bits
- 2's complement is assumed
- Instruction format: Each instruction is 32-bit wide, and consists of five fields: opcode, rd, rs, rt, and unused. The format is as follows:

opcode (4 bits)	rd (6 bits)	rs (6 bits)	rt (6 bits)	unused (10 bits)

Instruction	Symbol	Opcode	rd	rs	rt	Function
No operation	**NOP**	**0000**	×	×	×	**No operation**
Save PC	**SVPC rd, y**	**1111**	**rd**	**y**		**xrd ← PC+y**
Load	**LD rd, rs**	**1110**	**rd**	**rs**	×	**xrd ← M[xrs]**
Store	**ST rt, rs**	**0011**	×	**rs**	**rt**	**M[xrs] ← xrt**
Add	**ADD rd, rs, rt**	**0100**	**rd**	**rs**	**rt**	**xrd ← xrs + x rt**
Increment	**INC rd, rs, y**	**0101**	**rd**	**rs**	**y**	**x rd ← xrs + y**
Negate	**NEG rd, rs**	**0110**	**rd**	**rs**	×	**x rd ← - xrs**
Subtract	**SUB rd, rs, rt**	**0111**	**rd**	**rs**	**rt**	**x rd ← xrs – x rt**
Jump	**Jrs**	**1000**	×	**rs**	×	**PC ← xrs**
Branch if zero	**BRZrs**	**1001**	×	**rs**	×	**PC ← x rs, if Z = 1**
Jump memory	**JM rs**	**1010**	×	**rs**	×	**PC ← M[x rs]**
Branch if negative	**BRN rs**	**1011**	×	**rs**	×	**PC ← x rs, if N = 1**
MIN	**MIN, rd, rs, rt**	**0001**	**rd**	**rs**	**rt**	**See ***

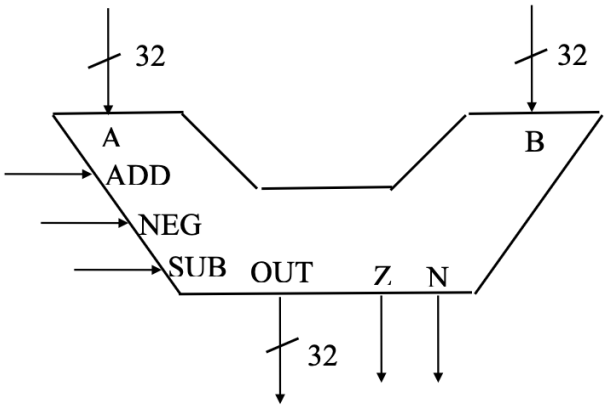
*x rd = **Min**$\{memory[xrs],\ memory[xrs+1], \ldots, memory[xrs+xrt-1]\}$
× **:** don't care

Instructions in the SCU ISA were used to write two versions of assembly program of finding the minimum in 'n' numbers described below.

(1) MIN = min$\{a_1, a_2, a_3, a_4, \ldots a_n\}$

The first version (software loop) does not use the MIN instruction and the second version (hardware loop) uses the MIN instruction.

ALU Block Diagram



ALU Truth Table

ADD	NEG	SUB	OUT	Operation
1	0	0	B+A	add
0	1	0	-B	2's complement
0	0	1	B-A	subtract
0	0	0	don't care	no operation
1	1	1	A	Pass A

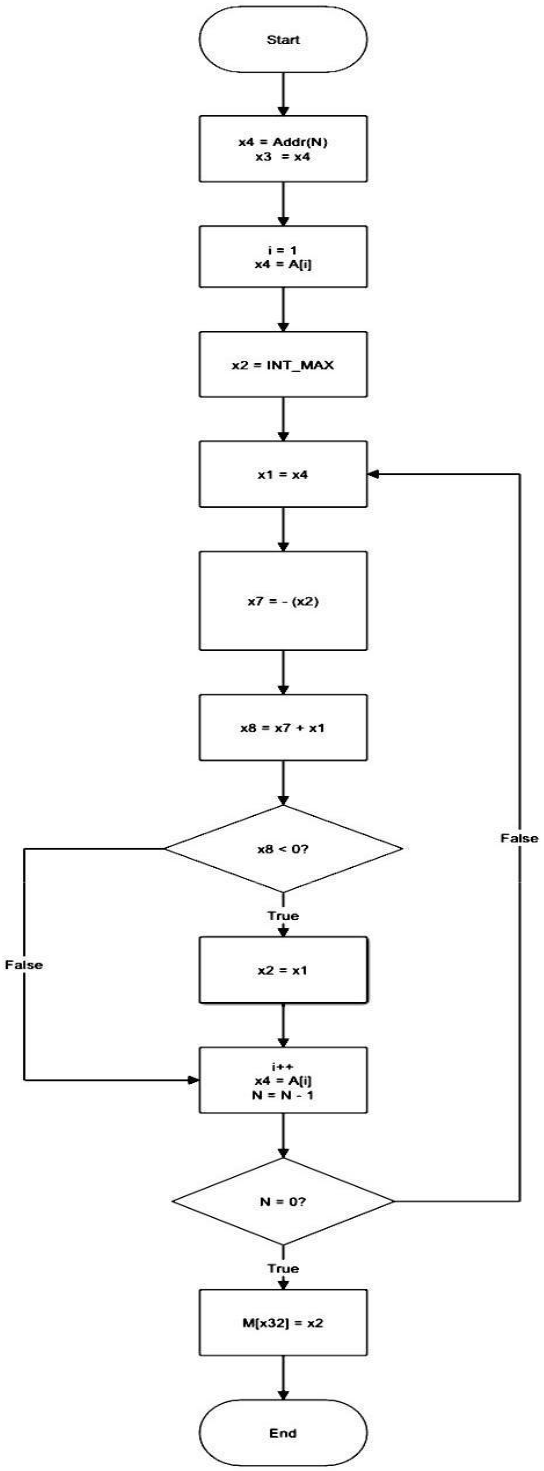
Z=1 if and only if OUT=0
N=1 if and only if OUT is negative.

Assumptions

1. The input array has at least one element i.e, the input array A is non-empty

2. The address of size of input array is stored in register x4
3. The input array is stored in memory location following x4. Memory is named M
4. When the execution of the program begins, the program counter (PC) points to the first instruction
5. Register x0 is reserved for the constant 0

Flow Chart



Assembly Code

Benchmark for Version 1: To find minimum number out of ‘n’ numbers without using label(s)

x1 - Current input value

x2 - Minimum value
x3 - Size of the input array, counter
x4 - Address of the size of the array
x6 - Start of the loop
x8 - Sum of current input value and minimum value
x10 - Position of array increment and counter decrement
x11 - Position of update minimum value
x12 - Position of storing the final minimum number
M[x32] - Memory location where final minimum value is stored

```
1. SVPC x40, x0                 // save the address of the first line of program in register x40
2. INC x10, x40, 0x39.          // calculate offset of line 13
3. INC x11, x40, 0x49           // calculate offset of line 17
4. INC x12, x40, 0x59           // calculate offset of line 19
5. LD x3, x4                    // load the size of input array to x3 from x4
6. INC x4, x4, 0x4              // move current position to the first element of the array
7. INC x2, x0, 0x7FFF FFFF      // x2 is initialized with INT_MAX
8. SVPC x6, 0x4                 // go to the start of the loop in memory stack
9. LD x1, x4                    // store current input value in x1
10. NEG x7, x2                  // negate the value in x2 and store in x7
11. ADD x8, x7, x1              // x8 = value in x7 + current input in x1
12. BRN x11                     // if min + current input < 0, go to update min
13. INC x4, x4, 0x4             // increment array position to point to next element
14. INC x3, x3, 0xFFFF FFFF     // decrement the counter
15. BRZ x12                     // if no more array elements are present, jump to the last line
16. J x6                        // else go to the start of the loop
17. INC x2, x1, 0x0             // update the minimum value in x2
18. J x10                       // jump to increment array position and decrement the counter
19. ST x2, x32                  // store the final minimum value in M[x32]
```

Equivalent C/C++ Code

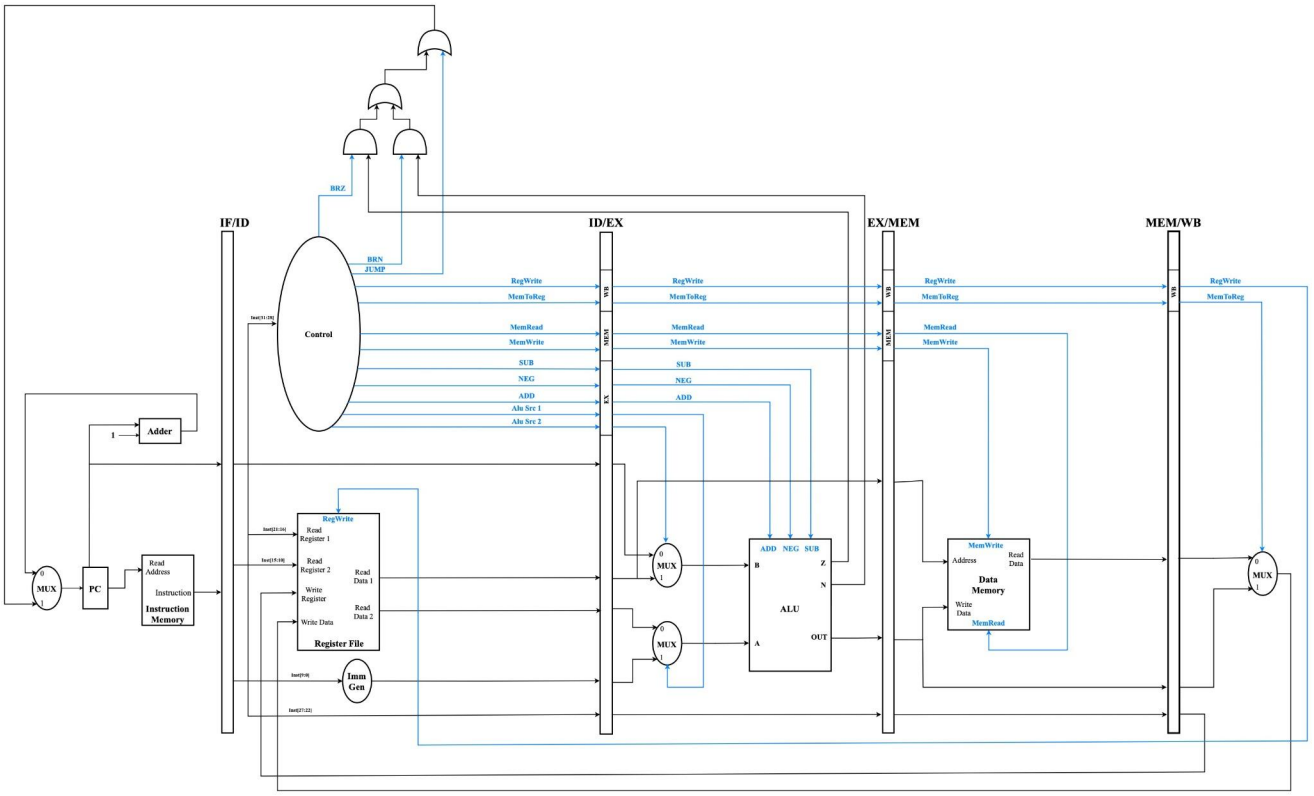
```
const int n = 5;
int arr[n];
int min = INT_MAX;

for (int i = 0; i < n; i++) {
  if (arr[i] < min){
    min = arr[i];
  }
}
return min;
```

Benchmark for Version 2: To find minimum number out of ‘n’ numbers using MIN instruction

x1 - Minimum value
x3 - Size of the input array, counter
x4 - Address of the size of the array
M[x32] - Memory location where final minimum value is stored

1. LD x3, x4 // load the size of array from x4 to x3
2. INC x4, x4, 0x4 // move current position to the first element of the array
3. MIN x1, x4, x3 // find minimum element of the array using MIN
4. ST x1, x32 // store the minimum value in M[x32]

Datapath & Control



Truth Table for Control Logic

Instruction	RegWrite	Mem Write	MemRead	ALU Src1	ALU Src2	BRN	BRZ	JUMP	ADD	SUB	NEG	MemTo Reg
Save PC	1	0	0	0	1	0	0	0	1	0	0	1
Load	1	0	1	XXX	XXX	0	0	0	0	0	0	0
Store	0	1	0	XXX	XXX	0	0	0	0	0	0	XXX
Add	1	0	0	0	1	0	0	0	1	0	0	1
Increment	1	0	0	1	1	0	0	0	1	0	0	1
Negate	1	0	0	XXX	1	0	0	0	0	0	1	1
Subtract	1	0	0	0	1	0	0	0	0	1	0	1
Jump	0	0	0	XXX	XXX	0	0	1	0	0	0	XXX
Branch if zero	0	0	0	XXX	XXX	0	1	0	0	0	0	XXX
Branch if negative	0	0	0	XXX	XXX	1	0	0	0	0	0	XXX

Emulation of RISC-V Instructions

For each RISC-V instruction mentioned, the equivalent instruction or a sequence of instructions in SCU ISA are listed

RISC-V	Equivalent SCU ISA
ADD x3, x1, x2	ADD x3, x1, x2
SUB x3, x1, x2	SUB x3, x1, x2
ADDI x3, x1, imm	INC x3, x1, imm
LW x3, imm(x1)	INC x1, x1, imm LD x3, x1
SW x3, imm(x1)	INC x1, x1, imm ST x3, x1

AND x2, x3, x4	SUB x2, x2, x2
	SVPC x10, 16
	ADD x3, x3, x4
	BRZ x10
	INC x2, x3, -1
	NOP

OR x2, x3, x4	SUB x2, x2, x2
	SVPC x10, 16
	ADD x3, x3, x4
	BRZ x10
	INC x2, x2, 1
	NOP

NOR x2, x3, x4	SUB x2, x2, x2
	INC x2, x2, 1
	SVPC x10, 16
	ADD x3, x3, x4
	BRZ x10
	INC x2, x2, -1
	NOP

ANDI x2, x3, imm	SUB x2, x2, x2 SVPC x10, 16 INC x3, x3, imm BRZ x10 INC x2, x2, -1 NOP
BEQ x1, x2, imm	SVPC x4, imm SUB x3, x1, x2 BRZ x4
JAL x3, imm	SVPC x3, 1 SVPC x4, imm J x4
JALR x3, x1, imm	SVPC x3, 1 INC x3, x1, imm J x4

Performance Analysis

Instruction Count

To prevent hazards, we can employ forwarding and flushing units. Thus, it is not necessary to include the NOP instruction in the code. After examining the assembly code, we note 9 lines of code (Lines 9 - 15) in the loop and, 12 lines of code outside the loop. Hence, the total instruction count will be -

Instruction Count = 7n + 12

Cycle Time

There are five stages in the pipeline, and we know that the delay times for the instruction memory, data memory, register file and ALU are 2ns, 1.5ns, and 2ns, respectively. Therefore, the clock cycle time should be set at 2ns.

CPI

To eliminate the control hazard, a flushing unit could be used. Thus, it will consume 1 more cycle because of the BRN instruction.

CPI = Total number of clock cycles/ Instruction count

Now, CPI = (K + N + 1 - 1) / N

Where,

K - number of stages

N - number of instructions

Hence, CPI = (5 + 7n + 12 + 1 – 1) / (7n + 12) = (7n + 17) / (7n + 12)

For example, if we consider an array of 5 elements, to calculate the minimum number the loop runs 5 times i.e. n = 5

Therefore, CPI = ((7 x 5) + 17) / ((7 x 5) + 12)

CPI = 1.106

When ‘n’ approaches infinity, CPI becomes almost equal to 1

Execution Time

Formula for calculating Execution time/CPU time is,

Execution Time = Instruction count x CPI x Clock Cycle Time

= (7n + 12) x [(7n + 17) / (7n + 12)] x 2ns

Now, for the example which we considered earlier,

Execution Time = ((7 x 5) + 12) x 1.106 x 2ns = 103.96ns