<u>CO224 – Computer Architecture</u>

<u>Lab 5 – Building a Simple Processor</u>

<u>Part 5 – Extended ISA – Report</u>

Group 27

E/19/170 - JAYAWARDHANA K.N.N.

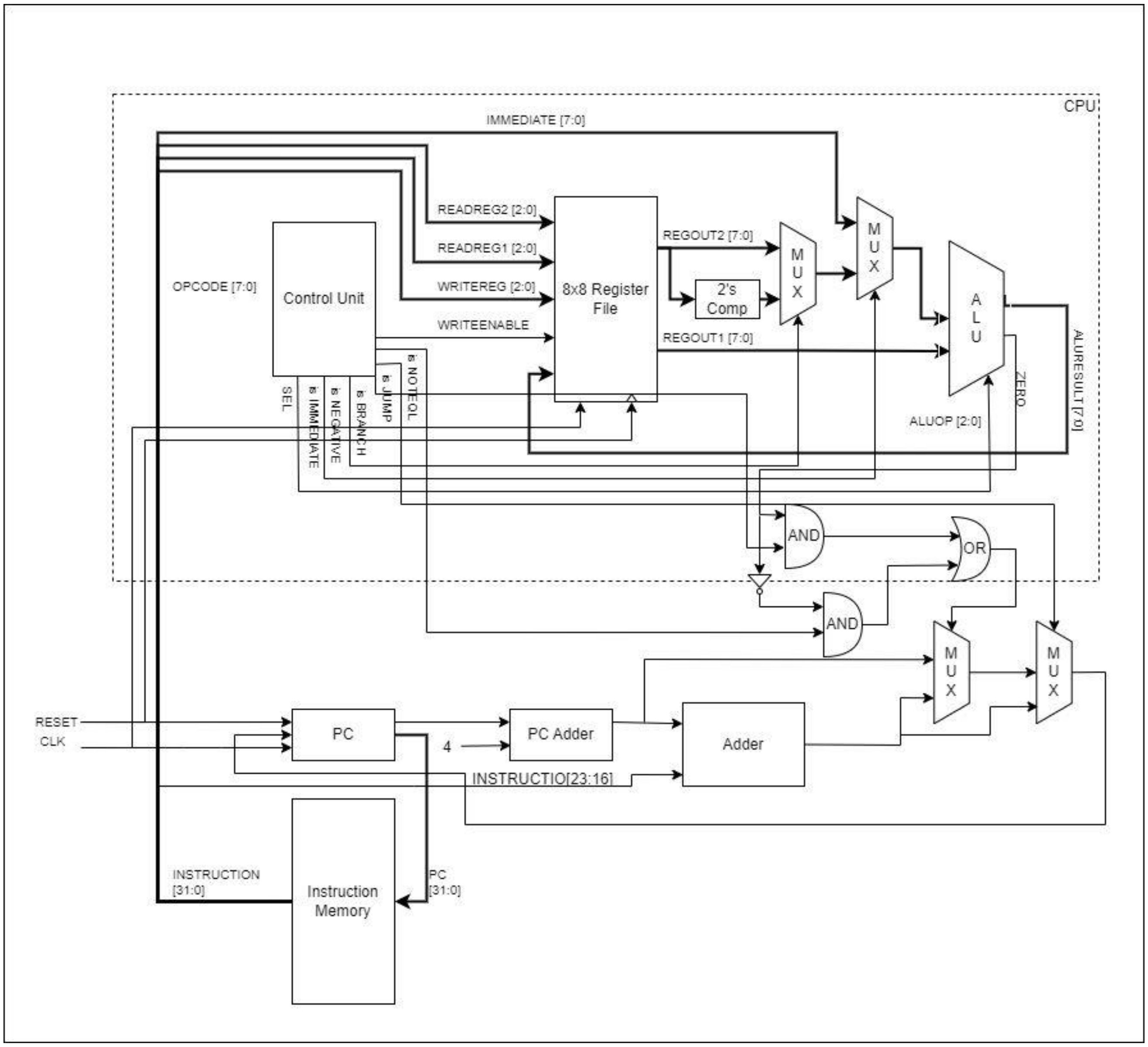
E/19/432 - WICKRAMAARACHCHI U.I.

The instruction set of the CPU was extended to include following five new instructions.

- **mult <r1> <r2> <r3>** : Multiplies the values in r2 and r3 and stores result in r1
- **sll <r1> <r2> <offset>** : Logically left shifts r2 by offset value and stores result in r1
- **srl <r1> <r2> <offset>** : Logically right shifts r2 by offset value and stores result in r1
- **sra <r1> <r2> <offset>** : Arithmetically right shifts r2 by offset value and stores result in r1
- **ror <r1> <r2> <offset>** : Rotates r2 right by offset value and stores result in r1
- **bne <offset> <r1> <r2>** : Branches based on offset if values in r1 and r2 are not equal

To implement these new instructions, a minor modification was made to the Branch/Jump hardware and some additional functional units were implemented within the ALU of the CPU as shown in the following modified block diagram.

Modified CPU Block Diagram



<u>ALU Functions</u>

SELECT	**Function**	**Description**	**Supported Instructions**		**Unit's Delay**
000	FORWARD	DATA2→RESULT	loadi, mov		#1
001	ADD	DATA1+DATA2→RESULT	add, sub, beq, bne		#2
010	AND	DATA1&DATA2→RESULT	and		#1
011	OR	DATA1|DATA2→RESULT	or		#1
100	SHIFT		**shift select**		#1
		(Logical Shift left) DATA1<<DATA2→RESULT	01	sll	
		(Logical Shift right) DATA1>>DATA2→RESULT	00	srl	
		(Arithmetic Shift right) DATA1>>>DATA2→RESULT	10	sra	
		Rotate Right DATA1 by DATA2 times	11	ror	
101	MULT	DATA1*DATA2→RESULT	mult		#2

CPU OPCODEs for Instruction Set

Instruction	OPCODE
loadi	00000000
mov	00000001
add	00000010
sub	00000011
and	00000100
or	00000101
j	00000110
beq	00000111
bne	00010000
mult	00010001
sll	00001100
srl	00001101
sra	00001110
ror	00001111

Test assembly code :

```
loadi 0 0x05   // r0 = 5
loadi 1 0x09   // r1 = 9
loadi 5 0x01   //r5 =1
or 7 0 1       //r7 = r0 | r1 = 13
mov 2 1        // r2 = r1=9
mov 6 1      //r6 = r1 = 9

beq 0x03 2 6 //branch to 3 ins forward

add 4 2 0      // r3 = r2 + r1 //skip
sub 4 3 1      // r4 = r3 - r1  //skip
add 3 2 5     // r3 = r2 +r5 = 9 +1  //skip

add 3 0 1     //5+9 =14

j 0x02        //jump 2 ins forward

loadi 0 0x75   // r0 = 0x75  //skip
loadi 1 0x92   // r1 = 0x92  //skip

loadi 0 0x02   //r0 =2
loadi 1 0x01   //r1 =1
and 5 1 0      // r5 = r1 & r0 0
or 7 0 1       // r7 = r1 | r2 3
sub 4 3 5     // r4 = 14-0

mult 4 3 0   //r4 = r3*r0 14*2 = 28
sll 4 1 0x02  // 1 << 2 = 4
srl 4 3 0x02  // 14 >>2 = 3
```

loadi 3 0x92 // r3 = -110

sra 4 3 0x02 // -28

loadi 0 0x7C // r0 = 124

ror 4 0 0x02 // 31

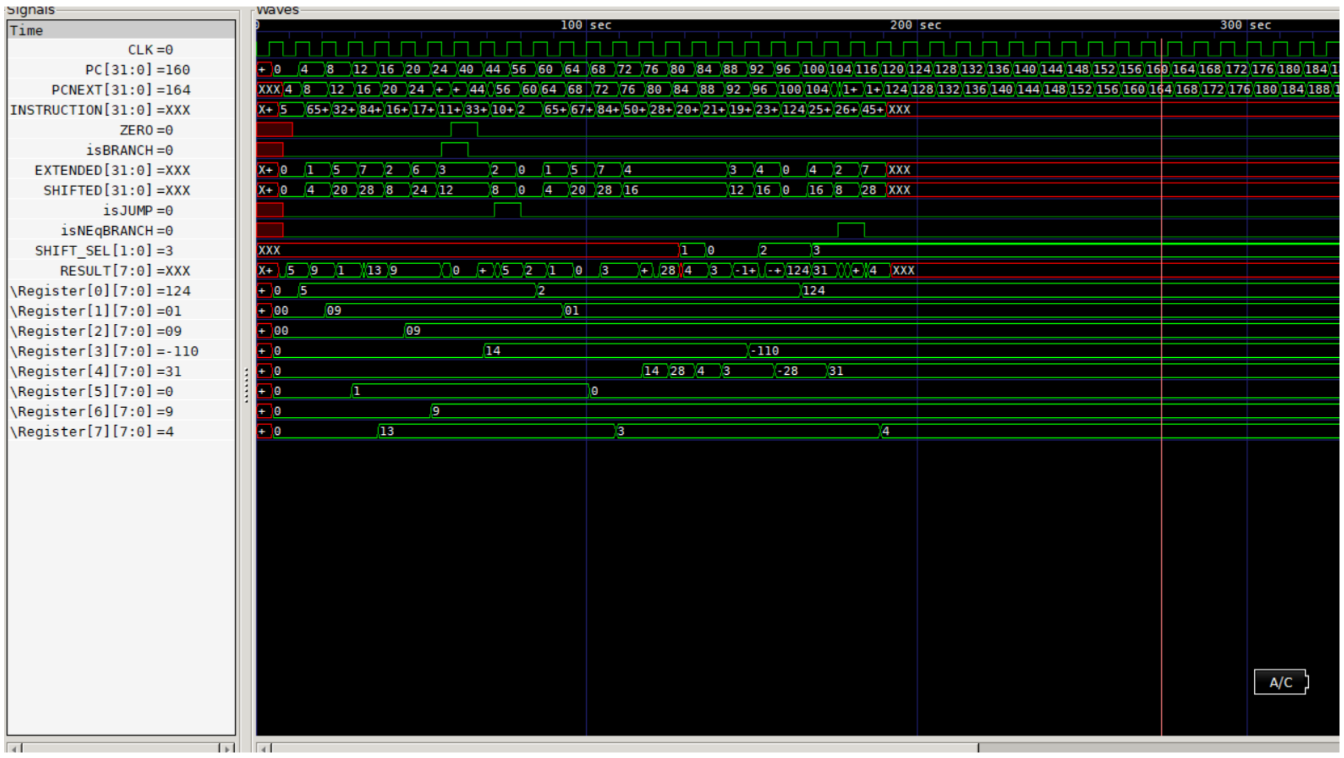
bne 0x02 0 3 // 124 != -110

loadi 7 0x01 //skip

loadi 7 0x02 //skip

loadi 7 0x04 //r7 = 4

Timing Diagram For test code :



```
Signals
Time
            CLK =0
      PC[31:0] =160
  PCNEXT[31:0] =164
INSTRUCTION[31:0] =XXX
           ZERO =0
       isBRANCH =0
  EXTENDED[31:0] =XXX
   SHIFTED[31:0] =XXX
         isJUMP =0
     isNEqBRANCH =0
   SHIFT_SEL[1:0] =3
     RESULT[7:0] =XXX
\Register[0][7:0] =124
\Register[1][7:0] =01
\Register[2][7:0] =09
\Register[3][7:0] =-110
\Register[4][7:0] =31
\Register[5][7:0] =0
\Register[6][7:0] =9
\Register[7][7:0] =4
```