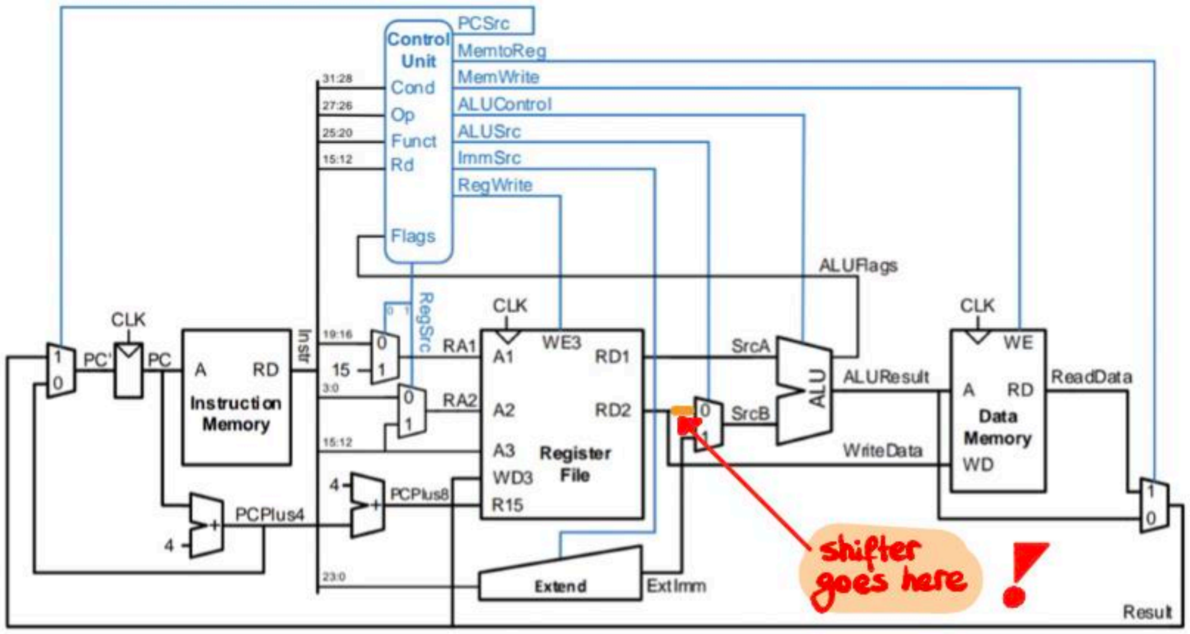


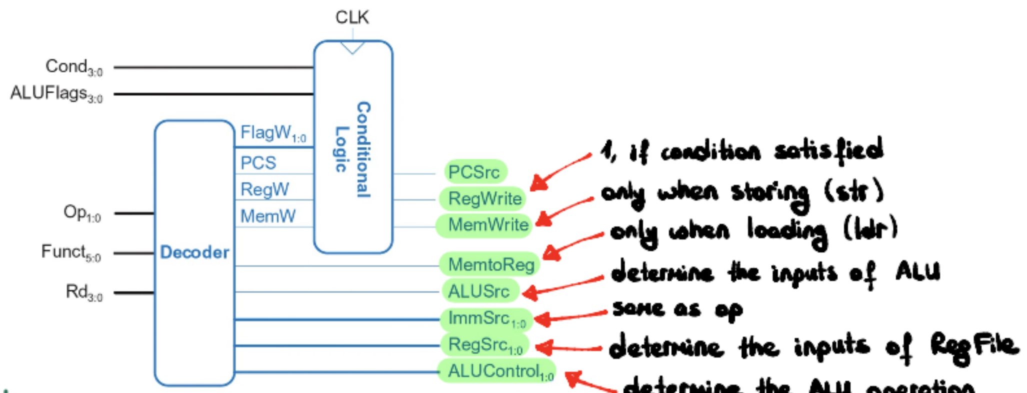
We only care for the Ⓩ flag!

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

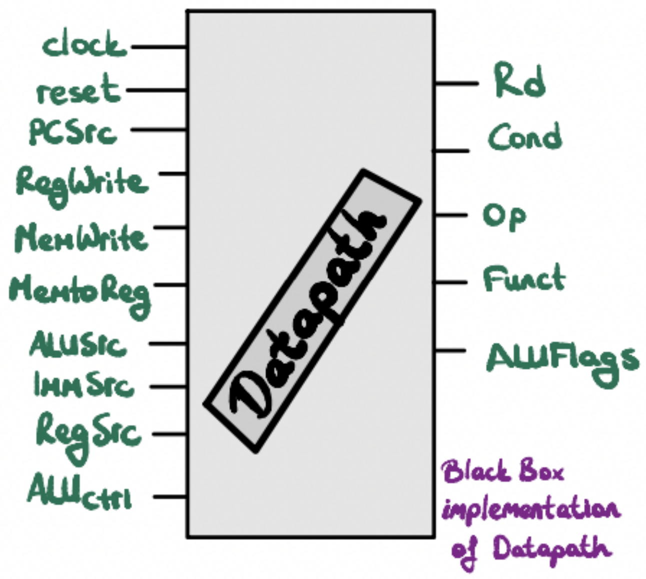
ISA to be implemented ...

Mnemonic	Name		Operation
ADD	Addition	add Rd,Rn,Rm	Rd← Rn + (Rm *sh* shamt5)
SUB	Subtraction	sub Rd,Rn,Rm	Rd← Rn - (Rm *sh* shamt5)
AND	Bitwise And	and Rd,Rn,Rm	Rd← Rn & (Rm *sh* shamt5)
ORR	Bitwise Or	orr Rd,Rn,Rm	Rd← Rn | (Rm *sh* shamt5)
MOV	Move to Register	mov Rd,Rm	Rd← (Rm *sh* shamt5)
STR	Store	str Rd,[Rn,imm12]	Mem[Rn + imm12] ← Rd
LDR	Load	ldr Rd,[Rn,imm12]	Rd ← Mem[Rn + imm12]
CMP	Compare	cmp Rd,Rn,Rm	set the flag if (Rn - Rm =0)
B	Branch	b imm24	PC ← imm24
BEQ	Branch if Equal	beq imm24	PC ← imm24 if flag = 1
BL	Branch with Link	bl imm24	PC ← imm24, R14 ← PC
BX	Branch and Exchange	bx Rm	PC ← Rm

We have to determine the control signals for each instruction.



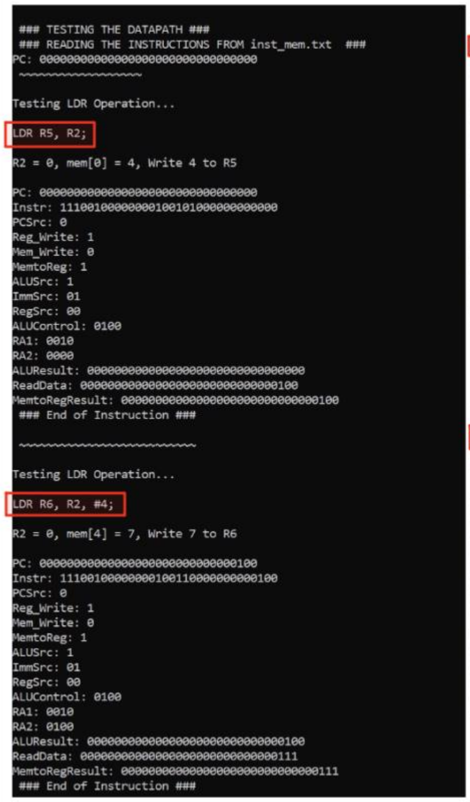
ISA	PCSrc	RegWrite	MemWrite	MemtoReg	ALUSrc	IMMSrc	RegSrc	ALUctrl
ADD	0	1 if cond	0	0	0	00	00	0100
SUB	0	1 if cond	0	0	0	00	00	0010
AND	0	1 if cond	0	0	0	00	00	0000
ORR	0	1 if cond	0	0	0	00	00	1100
MOV	0	1 if cond	0	0	0	00	0X	1101
STR	0	0	1 if cond	0	1	01	10	0100
LDR	0	1 if cond	0	1	1	01	X0	0100
CMP	0	0	0	0	0	XX	00	0010
B	1	0	0	0	1	10	XX	1101
BEQ	1	0	0	0	1	10	XX	1101
BL	1	0	0	0	1	10	X1	1101
BX	1	0	0	0	0	10	0X	1101



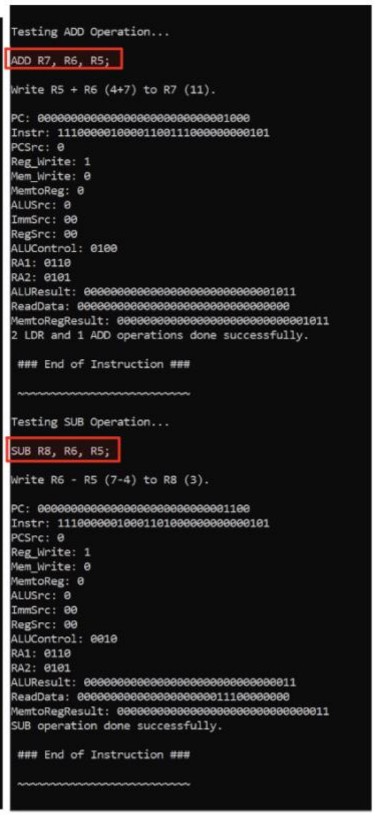
Datapath is tested with following instructions...

LDR R5, R2 ; R2=0, R5 ← mem[0] = 4
after this instruction, R5 = 4
LDR R6, R2, #4 ; R6 ← mem[0+4] = 7
after this instruction, R6 = 7
ADD R7, R6, R5 ; R7 ← R5 + R6
after this instruction, R7 = 11
SUB R8, R6, R5 ; R8 ← R6 - R5
after this instruction, R8 = 3
STR R8, R2, #12 ; R8 → mem [0+12]
at memory loc. 12, 3 is written
LDR R0, R2, #12 ; R0 ← mem[0+12] = 3
after this instruction, R0 = 3
ORR R3, R0, R5 ; R3 ← (0011) | (0100)
after this instruction, R3 = 0111
AND R4, R3, R0, #2 ; R4 ← (0111) & (1100)
after this instruction, R4 = 0100
MOV R1, R4, #2 ; R1 ← (4×2^{2}) ← shift
after this instruction, R1 = 10000
B #64 ; PC ← 64
after this instruction, PC = 64

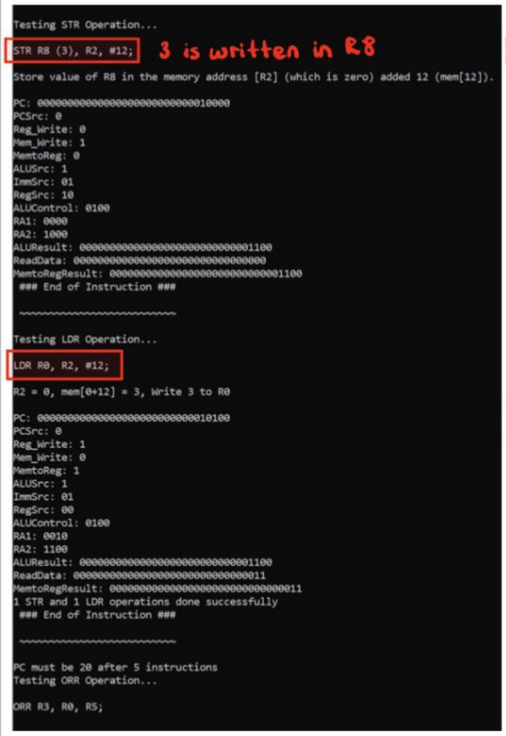
Following pages have cocotb terminal results of these instructions for the datapath.



```
 ### TESTING THE DATAPATH ###
 ### READING THE INSTRUCTIONS FROM inst_mem.txt  ###
PC: 00000000000000000000000000000000
 ~~~~~~~~~~~~~~~~~~

Testing LDR Operation...

LDR R5, R2;

R2 = 0, mem[0] = 4, Write 4 to R5

PC: 00000000000000000000000000000000
Instr: 11100100000000010010100000000000
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 1
ALUSrc: 1
ImmSrc: 01
RegSrc: 00
ALUControl: 0100
RA1: 0010
RA2: 0000
ALUResult: 00000000000000000000000000000000
ReadData: 00000000000000000000000000000100
MemtoRegResult: 00000000000000000000000000000100
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing LDR Operation...

LDR R6, R2, #4;

R2 = 0, mem[4] = 7, Write 7 to R6

PC: 00000000000000000000000000000100
Instr: 11100100000000010011000000000100
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 1
ALUSrc: 1
ImmSrc: 01
RegSrc: 00
ALUControl: 0100
RA1: 0010
RA2: 0100
ALUResult: 00000000000000000000000000000100
ReadData: 00000000000000000000000000000111
MemtoRegResult: 00000000000000000000000000000111
 ### End of Instruction ###
```



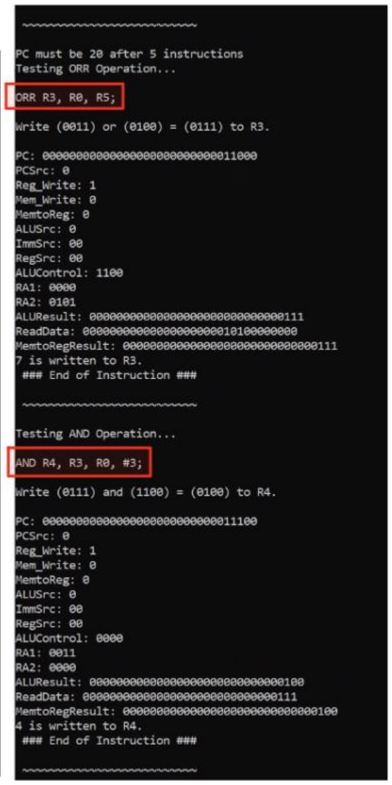
```
Testing ADD Operation...

ADD R7, R6, R5;

Write R5 + R6 (4+7) to R7 (11).

PC: 00000000000000000000000000001000
Instr: 11100000100001100111000000000101
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 0100
RA1: 0110
RA2: 0101
ALUResult: 00000000000000000000000000001011
ReadData: 00000000000000000000000000000000
MemtoRegResult: 00000000000000000000000000001011
2 LDR and 1 ADD operations done successfully.

 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing SUB Operation...

SUB R8, R6, R5;

Write R6 - R5 (7-4) to R8 (3).

PC: 00000000000000000000000000001100
Instr: 11100000010001101000000000000101
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 0010
RA1: 0110
RA2: 0101
ALUResult: 00000000000000000000000000000011
ReadData: 00000000000000000000011100000000
MemtoRegResult: 00000000000000000000000000000011
SUB operation done successfully.

 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~
```



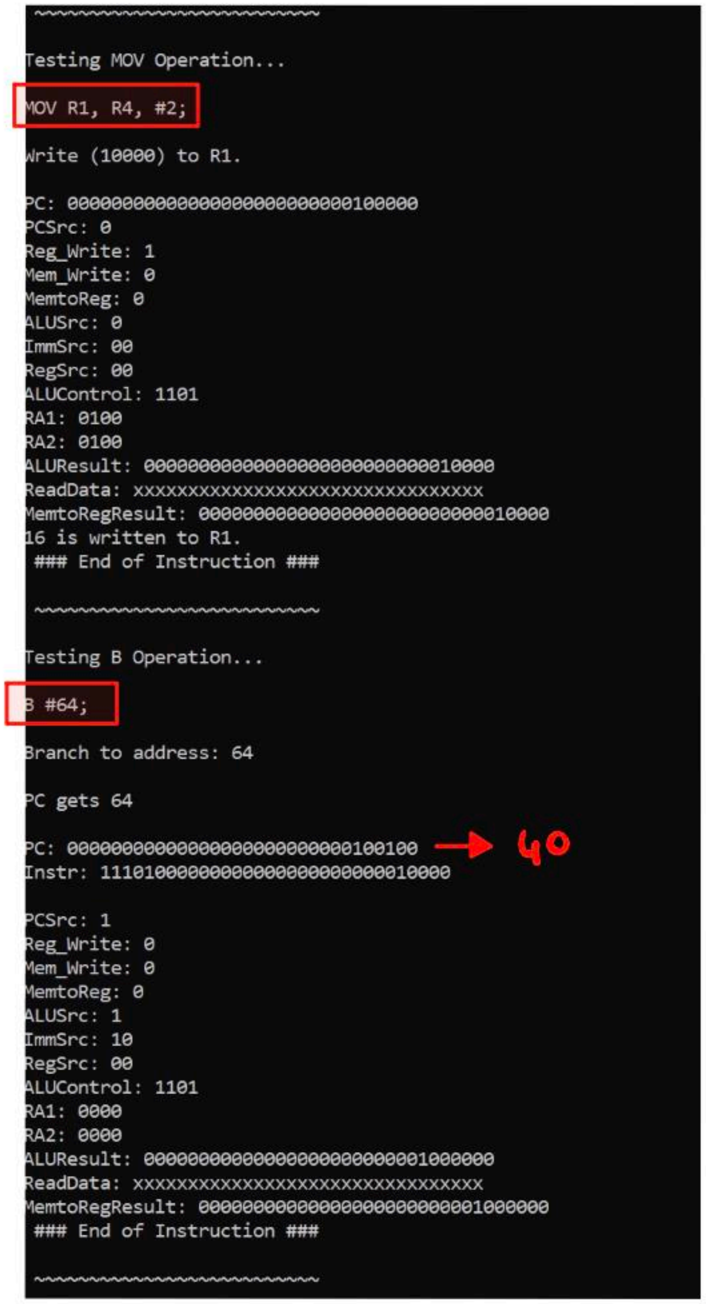
```
Testing STR Operation...

STR R8 (3), R2, #12;
```

3 is written in R8

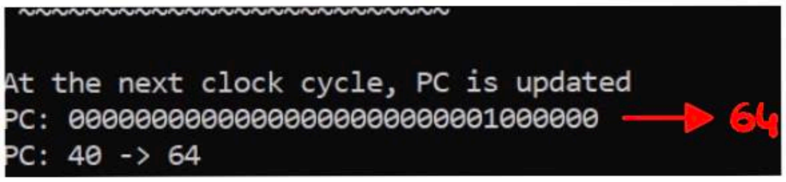
```
Store value of R8 in the memory address [R2] (which is zero) added 12 (mem[12]).

PC: 00000000000000000000000000010000
PCSrc: 0
Reg_Write: 0
Mem_Write: 1
MemtoReg: 0
ALUSrc: 1
ImmSrc: 01
RegSrc: 10
ALUControl: 0100
RA1: 0000
RA2: 1000
ALUResult: 00000000000000000000000000001100
ReadData: 00000000000000000000000000000000
MemtoRegResult: 00000000000000000000000000001100
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing LDR Operation...

LDR R0, R2, #12;

R2 = 0, mem[0+12] = 3, Write 3 to R0

PC: 00000000000000000000000000010100
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 1
ALUSrc: 1
ImmSrc: 01
RegSrc: 00
ALUControl: 0100
RA1: 0010
RA2: 1100
ALUResult: 00000000000000000000000000001100
ReadData: 00000000000000000000000000000011
MemtoRegResult: 00000000000000000000000000000011
1 STR and 1 LDR operations done successfully
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

PC must be 20 after 5 instructions
Testing ORR Operation...

ORR R3, R0, R5;
```



```
 ~~~~~~~~~~~~~~~~~~~~~~~~~~

PC must be 20 after 5 instructions
Testing ORR Operation...

ORR R3, R0, R5;

Write (0011) or (0100) = (0111) to R3.

PC: 00000000000000000000000000011000
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 1100
RA1: 0000
RA2: 0101
ALUResult: 00000000000000000000000000000111
ReadData: 00000000000000000000010100000000
MemtoRegResult: 00000000000000000000000000000111
7 is written to R3.
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing AND Operation...

AND R4, R3, R0, #3;

Write (0111) and (1100) = (0100) to R4.

PC: 00000000000000000000000000011100
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 0000
RA1: 0011
RA2: 0000
ALUResult: 00000000000000000000000000000100
ReadData: 00000000000000000000000000000111
MemtoRegResult: 00000000000000000000000000000100
4 is written to R4.
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~
```

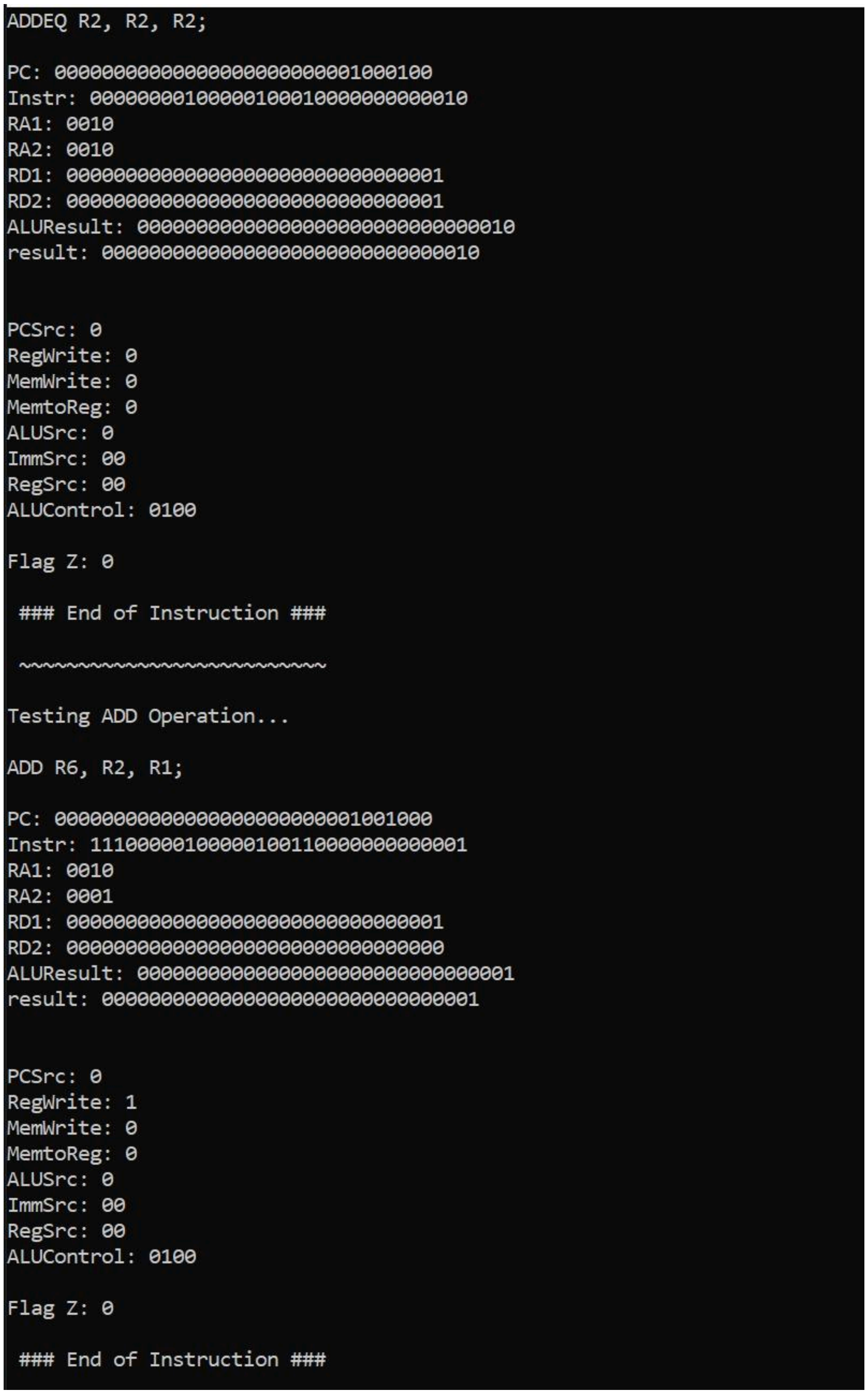


```
 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing MOV Operation...

MOV R1, R4, #2;

Write (10000) to R1.

PC: 00000000000000000000000000100000
PCSrc: 0
Reg_Write: 1
Mem_Write: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 1101
RA1: 0100
RA2: 0100
ALUResult: 00000000000000000000000000010000
ReadData: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MemtoRegResult: 00000000000000000000000000010000
16 is written to R1.
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing B Operation...

B #64;

Branch to address: 64

PC gets 64

PC: 00000000000000000000000000100100  → 40
Instr: 11101000000000000000000000010000

PCSrc: 1
Reg_Write: 0
Mem_Write: 0
MemtoReg: 0
ALUSrc: 1
ImmSrc: 10
RegSrc: 00
ALUControl: 1101
RA1: 0000
RA2: 0000
ALUResult: 00000000000000000000000001000000
ReadData: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MemtoRegResult: 00000000000000000000000001000000
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~
```



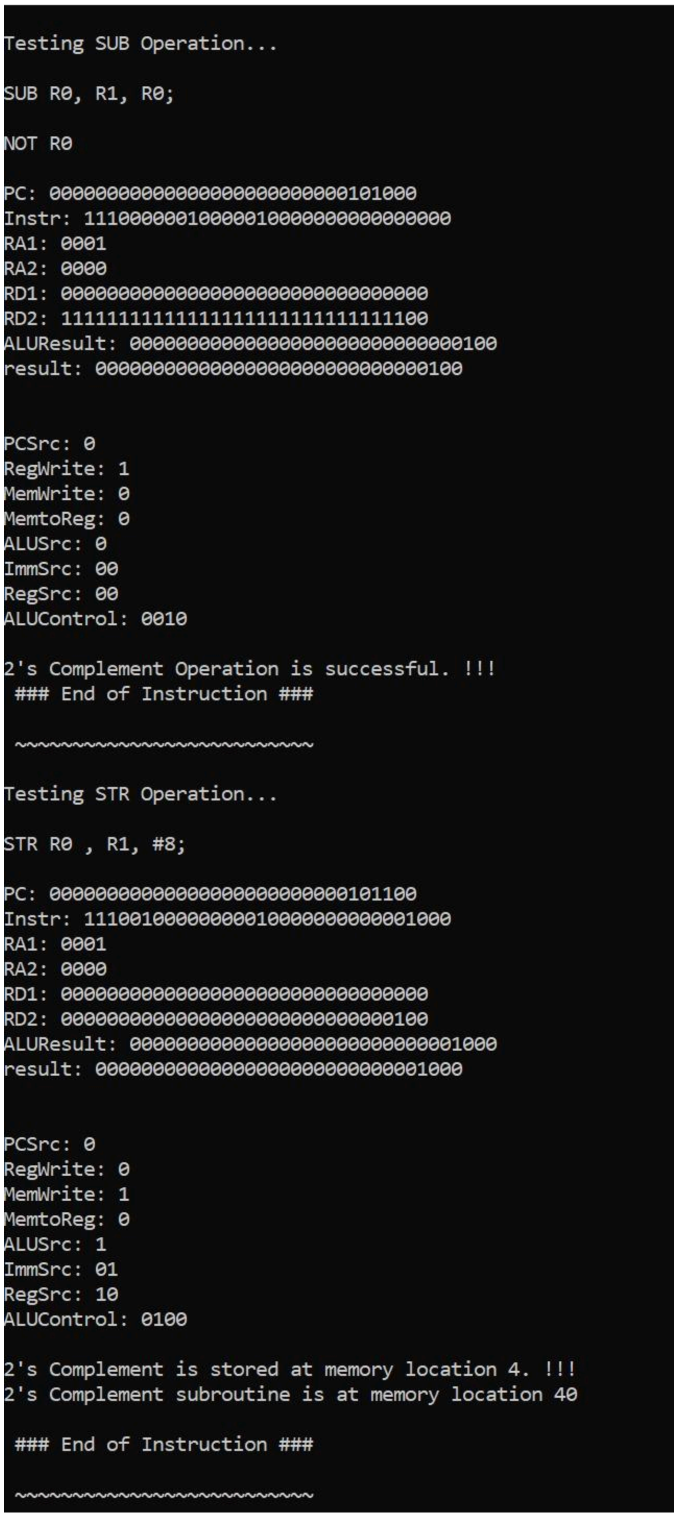
```
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

At the next clock cycle, PC is updated
PC: 00000000000000000000000001000000  → 64
PC: 40 -> 64
```

Result is not written in R2 since condition is not satisfied.



```
ADDEQ R2, R2, R2;

PC: 00000000000000000000000001000100
Instr: 00000000100000100010000000000010
RA1: 0010
RA2: 0010
RD1: 00000000000000000000000000000001
RD2: 00000000000000000000000000000001
ALUResult: 00000000000000000000000000000010
result: 00000000000000000000000000000010


PCSrc: 0
RegWrite: 0
MemWrite: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 0100

Flag Z: 0

 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing ADD Operation...

ADD R6, R2, R1;

PC: 00000000000000000000000001001000
Instr: 11100000100000100110000000000001
RA1: 0010
RA2: 0001
RD1: 00000000000000000000000000000001
RD2: 00000000000000000000000000000000
ALUResult: 00000000000000000000000000000001
result: 00000000000000000000000000000001


PCSrc: 0
RegWrite: 1
MemWrite: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 0100

Flag Z: 0

 ### End of Instruction ###
```

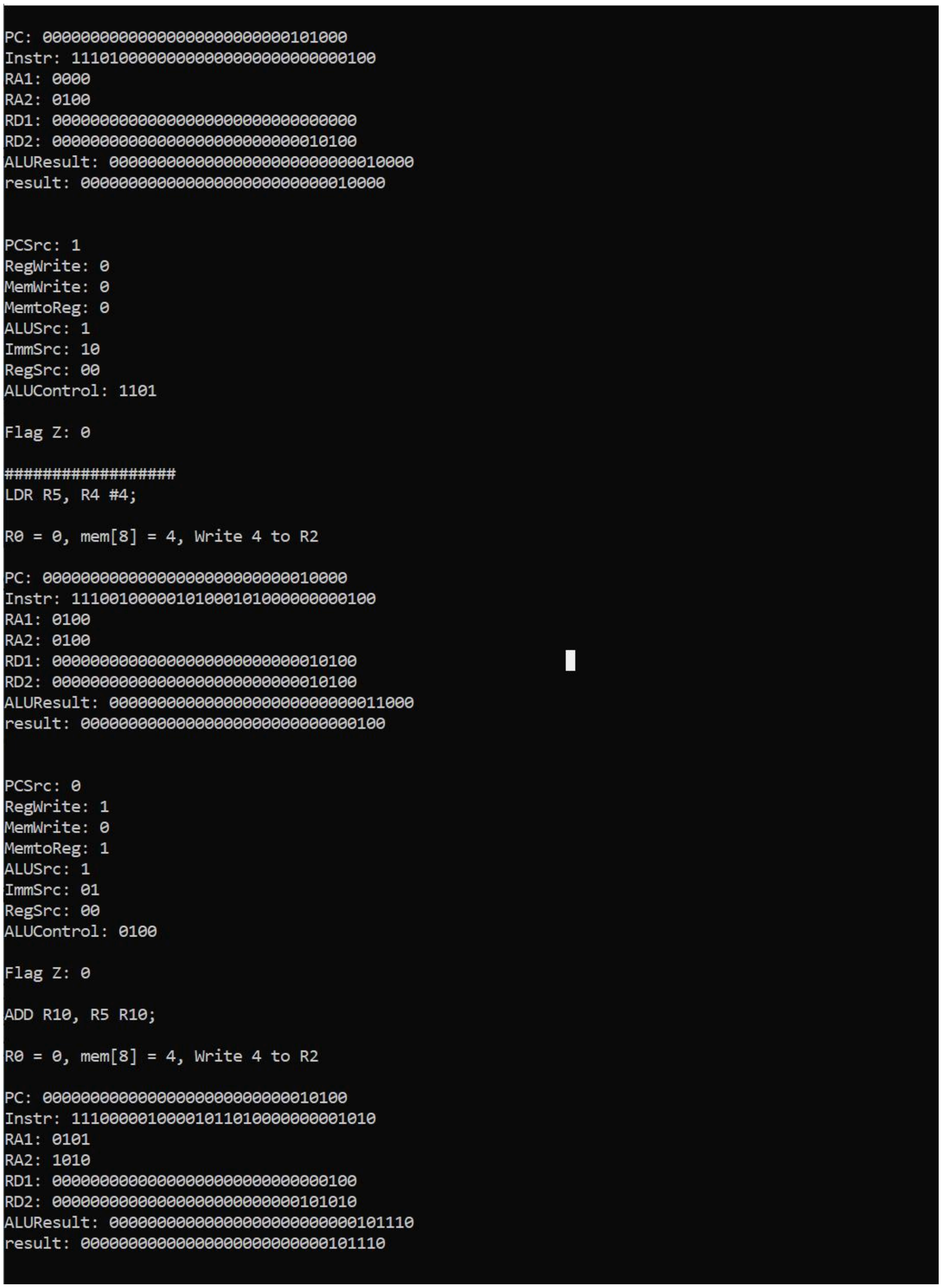
2’s Complement Subroutine is at memory location 40.



```
Testing SUB Operation...

SUB R0, R1, R0;

NOT R0

PC: 00000000000000000000000000101000
Instr: 11100000010000010000000000000000
RA1: 0001
RA2: 0000
RD1: 00000000000000000000000000000000
RD2: 11111111111111111111111111111100
ALUResult: 00000000000000000000000000000100
result: 00000000000000000000000000000100


PCSrc: 0
RegWrite: 1
MemWrite: 0
MemtoReg: 0
ALUSrc: 0
ImmSrc: 00
RegSrc: 00
ALUControl: 0010

2's Complement Operation is successful. !!!
 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~

Testing STR Operation...

STR R0 , R1, #8;

PC: 00000000000000000000000000101100
Instr: 11100100000000010000000000001000
RA1: 0001
RA2: 0000
RD1: 00000000000000000000000000000000
RD2: 00000000000000000000000000000100
ALUResult: 00000000000000000000000000001000
result: 00000000000000000000000000001000


PCSrc: 0
RegWrite: 0
MemWrite: 1
MemtoReg: 0
ALUSrc: 1
ImmSrc: 01
RegSrc: 10
ALUControl: 0100

2's Complement is stored at memory location 4. !!!
2's Complement subroutine is at memory location 40

 ### End of Instruction ###

 ~~~~~~~~~~~~~~~~~~~~~~~~~~
```

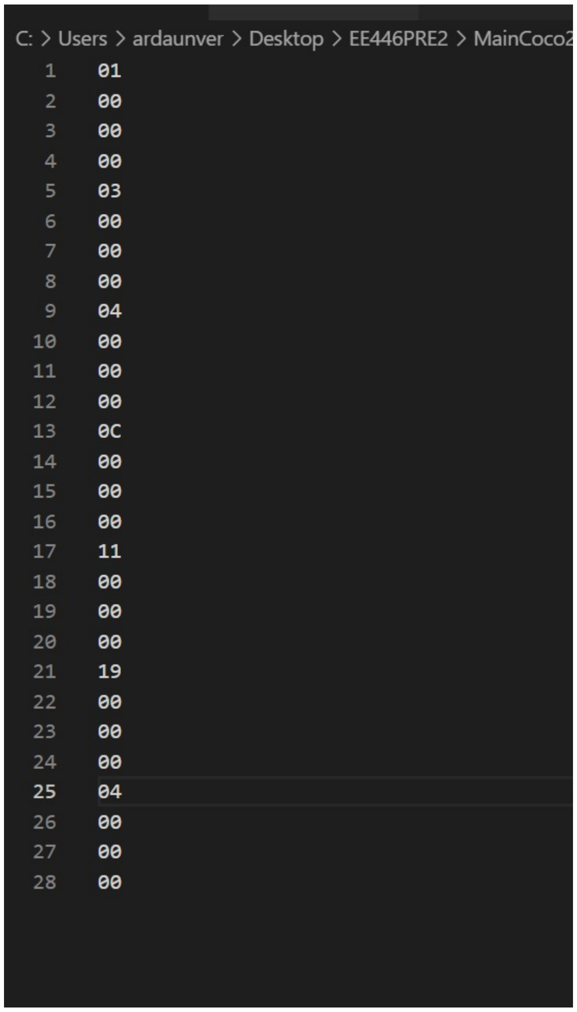
You can see main_COCO file for more examples.

Adding the numbers, 17,25,4…
And, obtained 46 as result after 3 loops.



```
PC: 00000000000000000000000000101000
Instr: 11101000000000000000000000000100
RA1: 0000
RA2: 0100
RD1: 00000000000000000000000000000000
RD2: 00000000000000000000000000010100
ALUResult: 00000000000000000000000000010000
result: 00000000000000000000000000010000


PCSrc: 1
RegWrite: 0
MemWrite: 0
MemtoReg: 0
ALUSrc: 1
ImmSrc: 10
RegSrc: 00
ALUControl: 1101

Flag Z: 0

##################
LDR R5, R4 #4;

R0 = 0, mem[8] = 4, Write 4 to R2

PC: 00000000000000000000000000010000
Instr: 11100100000101000101000000000100
RA1: 0100
RA2: 0100
RD1: 00000000000000000000000000010100
RD2: 00000000000000000000000000010100
ALUResult: 00000000000000000000000000011000
result: 00000000000000000000000000000100


PCSrc: 0
RegWrite: 1
MemWrite: 0
MemtoReg: 1
ALUSrc: 1
ImmSrc: 01
RegSrc: 00
ALUControl: 0100

Flag Z: 0

ADD R10, R5 R10;

R0 = 0, mem[8] = 4, Write 4 to R2

PC: 00000000000000000000000000010100
Instr: 11100000100001011010000000001010
RA1: 0101
RA2: 1010
RD1: 00000000000000000000000000000100
RD2: 00000000000000000000000000101010
ALUResult: 00000000000000000000000000101110
result: 00000000000000000000000000101110
```

Starting the sum from the address 16.



C: > Users > ardaunver > Desktop > EE446PRE2 > MainCoco2

```
01
00
00
00
03
00
00
00
04
00
00
00
0C
00
00
00
11
00
00
00
19
00
00
00
04
00
00
00
```