ASSIGNMENT 2

- ABHISHEK SHARMA
- ROLL NO. 2020101050
- 2020101050%4=2, 2+1 = 3
- TASK 3 IS ASSIGNED

INSTRUCTION 3
cxchng[XX] rA , rB :- Swap the values in rA and rB. The swap only happens if the condition [XX] is satisfied. Here [XX] can be any one of the jump conditions.
Assumption: for icode that is cxchng I take it as ‘10’
And for ifun that are jumping conditions [XX], they are given as

Task1:

cxchng	10(a)	0
cxchng[le]	10(a)	1
cxchng[l]	10(a)	2
cxchng[e]	10(a)	3
cxchng[ne]	10(a)	4
cxchng[ge]	10(a)	5

cxchange[g]	10(a)	6

Instruction set:

10	fn	Ra	Rb

Ra and Rb are one of the registers given in the book. Every box is 4 bits long, the other operations such as rrmovq etc. will have the same structure as given in the book.

TASK2:

#Excution begins at address 0

```
  .pos
  irmovq stack,%rsp
  call main
  halt

   .align 8
         array:
.quad 0x01
.quad 0x0d
.quad 0x04
.quad 0x0b
.quad 0x02
.quad 0x05
.quad 0x12
```

```
.quad 0x03

main:
  irmovq array,%rdi
  irmovq $8,%rsi
  call loop
  ret

loop:
   irmovq $8,%r8   #for array accessing
   irmovq $56,%r13
   rrmovq %rdi,%rbx   #copying the array address for left right traveesal
   addq %r13,%rbx

   #

  #in every check
  #first step --> get the left most element
  #2nd step---->get the right most element
  #3rd step--->using the new instruction to swap the element if the right one instruction greater than the left element with
exchng[xx]
   #4th step----->storing again values in rdi
  #5th step------> same in rbx
  #6thth step---->increment i (the left position)
  #7th step----->decrement the right most position
```

```
#1check
   mrmovq (%rdi),%r11
   mrmovq (%rbx),%r12
   cxchngg %r12,%r11
   rmmovq %r11,(%rdi)
   rmmovq %r12,(%rbx)
   addq %r8,%rdi
   subq %r8,%rbx

#2check
   mrmovq (%rdi),%r11
   mrmovq (%rbx),%r12
   cxchngg %r12,%r11
   rmmovq %r11,(%rdi)
   rmmovq %r12,(%rbx)
   addq %r8,%rdi
   subq %r8,%rbx

 #3check
   mrmovq (%rdi),%r11
   mrmovq (%rbx),%r12
   cxchngg %r12,%r11
   rmmovq %r11,(%rdi)
   rmmovq %r12,(%rbx)
   addq %r8,%rdi
```

```
   subq %r8,%rbx

#4check
   mrmovq (%rdi),%r11
   mrmovq (%rbx),%r12
   cxchngg %r12,%r11
   rmmovq %r11,(%rdi)
   rmmovq %r12,(%rbx)


ret
  # Stack starts here and grows to lower addresses
   .pos 0x200

stack:
```

TASK3:

```
#Excution begins at address 0
0x000:         .pos
0x000: 30f40002000000000000 irmovq stack,%rsp
0x00a: 805800000000000000   call main
```

```
0x013: 00                    halt

0x014: .align 8
0x018:         array:
0x018: 01     .quad 0x01
0x020: 0d     .quad 0x0d
0x028: 04     .quad 0x04
0x030: 0b     .quad 0x0b
0x038: 02     .quad 0x02
0x040: 05     .quad 0x05
0x048: 12     .quad 0x12
0x050: 03      .quad 0x03

0x058:  main:
0x058:30f71800000000000000          irmovq array,%rdi
0x062:30f71800000000000000          irmovq $8,%rsi
0x06c:807600000000000000            call loop
0x075:90                           ret

0x076: loop:
0x076:30f80800000000000000          irmovq $8,%r8   #for array accessing

0x080:30fd3800000000000000          irmovq $56,%r13
0x08a:2073                          rrmovq %rdi,%rbx
0x094:60d3                          addq %r13,%rbx
```

```
0x096:  #1check
0x096: 50b70000000000000000        mrmovq (%rdi),%r11
0x0a0: 50c30000000000000000        mrmovq (%rbx),%r12
0x0aa: a6cb                        cxchngg %r12,%r11
0x0ac: 40b70000000000000000        rmmovq %r11,(%rdi)
0x0b6: 40c30000000000000000        rmmovq %r12,(%rbx)
0x0c0:6087                         addq %r8,%rdi
0x0c2:6183                        subq %r8,%rbx


0x0c4: #2check
0x0c4: 50b70000000000000000   mrmovq (%rdi),%r11
0x0ce: 50c30000000000000000   mrmovq (%rbx),%r12
0x0d8: a6cb                  cxchngg %r12,%r11
0x0da: 40b70000000000000000   rmmovq %r11,(%rdi)
0x0e4: 40c30000000000000000   rmmovq %r12,(%rbx)
0x0ee: 6087                   addq %r8,%rdi
0x0f0: 6183                  subq %r8,%rbx


0x0f2: #3check
0x0f2:50b70000000000000000   mrmovq (%rdi),%r11
```

```
0x0fc:50c30000000000000000   mrmovq (%rbx),%r12
0x106:a6cb                   cxchngg %r12,%r11
0x108:40b70000000000000000   rmmovq %r11,(%rdi)
0x114:40c30000000000000000   rmmovq %r12,(%rbx)
0x11e:6087                   addq %r8,%rdi
0x120:6183                   subq %r8,%rbx

0x122: #4check
0x122: 50b70000000000000000   mrmovq (%rdi),%r11
0x12c: 50c30000000000000000   mrmovq (%rbx),%r12
0x136: a6cb                   cxchngg %r12,%r11
0x138: 40b70000000000000000   rmmovq %r11,(%rdi)
0x142: 40c30000000000000000   rmmovq %r12,(%rbx)


0x14c:90 ret
  # Stack starts here and grows to lower addresses
   .pos 0x200

stack:
```

TASK4:

The first instance of new instruction appears at 0x00aa, so we will proceed to task 4 from this address to show different stages.In the execute stage two ALUs which are giving results as valE1 and valE2 . The two consequences are discussed in the table below. According to the first instance we are taking the values of R11 and R12 accordingly:-

STAGE	cxchng[g] R11,R12
Fetch	Icode:ifun ← **0x00aa(PC)** R11:R12 ← **0x00ab(PC+1)** valP ← **0x00ac(PC+2)**
Decode	valA ← R[R11] **0x01** valB ← R[R12] **0x03**

Execute	if(valB > valA) **(0x03>0x01)**
	cxchngg valA, ValB works
	valE1 ← valB **0x03**
	valE2 ← valA **0x01**
	Else
	valE1 ← valA
	valE2 ← valB
	
	if(valB<valA) then CC→ 1 for sign flag

Memory	
Write Back	If swapping happens R[R11] ← valE1 R[R12] ← valE2 Else R[R11] ← valE2 R[R12] ← valE1
PC Update	PC ← valP **0x00ac(PC+2)**

TASK5:

In r11 and r12 the values are changing as the elements of array which are represented as A[ ] ,

cycles	PC	CC	%rsp	%rdi	%rsi	%r8	%r9	%r10	%r11	%r12	%r13	%rbx
1	0x000	(0,0,0)	0x200									

2	0x00a	(0,0,0)	0x1f8									
3	0x058	(0,0,0)		0x018								
4	0x062	(0,0,0)			0x008							
5	0x06c	(0,0,0)	0x1f0									
6	0x076	(0,0,0)				0x008						
7	0x080	(0,0,0)									0x038	
8	0x08a	(0,0,0)										0x018
9	0x094	(0,0,0)										0x050
10	0x096	(0,0,0)							A[0x018]			
11	0x0a0	(0,0,0)								A[0x050]		
12	0x0aa	(0,0,0)							If it swaps, the values interchange , otherwise same	If it swaps, the values interchange , otherwise same		
13	0x0ac	(0,0,0)		A[0x018]								
14	0x0b6	(0,0,0)										A[0x050]

15	0x0c0	(0,0,0)		0x020								
16	0x0c2	(0,0,0)										0x0048
17	0x0c4	(0,0,0)							A[0x020]			
18	0x0ce	(0,0,0)								A[0x0048]		
19	0x0d8	(0,0,0)							If it swaps, the values interchange , otherwise same	If it swaps, the values interchange , otherwise same		
20	0x0da	(0,0,0)		A[0x028]								

TASK6:

So , we have to calculate the total no. of cycles required to run the program.
So before main function the program has :

```
.pos
irmovq stack,%rsp   (1)
call main           (1)
Halt


#Calling main function
main:
irmovq array,%rdi    (1)
irmovq $8,%rsi       (1)
call loop            (1)
ret

loop:
irmovq $8,%r8        (1)
irmovq $56,%r13      (1)
rrmovq %rdi,%rbx     (1)
addq %r13,%rbx       (3) + (1)  [ for add instruction the the r13 and rbx has to be written back from previous
                                  Instruction, and then fetching will take place , so 3 bubbles will be there]
```

The 1,2,3 check or loop parts are

mrmovq (%rdi),%r11 (1)
mrmovq (%rbx),%r12 (1)
cxchngg %r12,%r11 (3) + (1) [for swap instruction assuming it will complete in one clock cycle but for no Dependencies we will give three nopes such that fetch and write back will come In order]

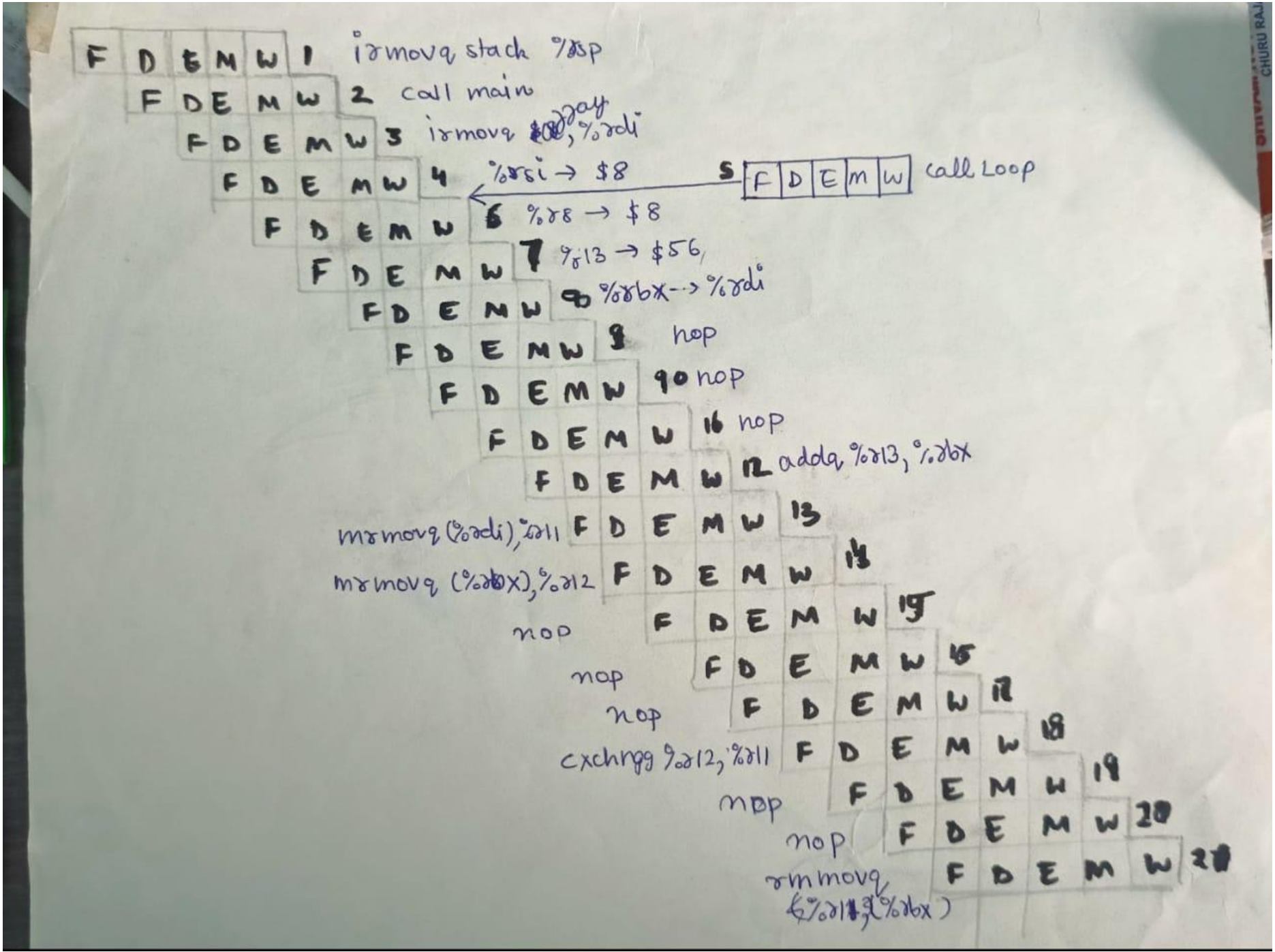
rmmovq %r11,(%rdi) (2) + (1) [same for this also after swapping values again stores in registers so for that we Have to wait till memory write stage]

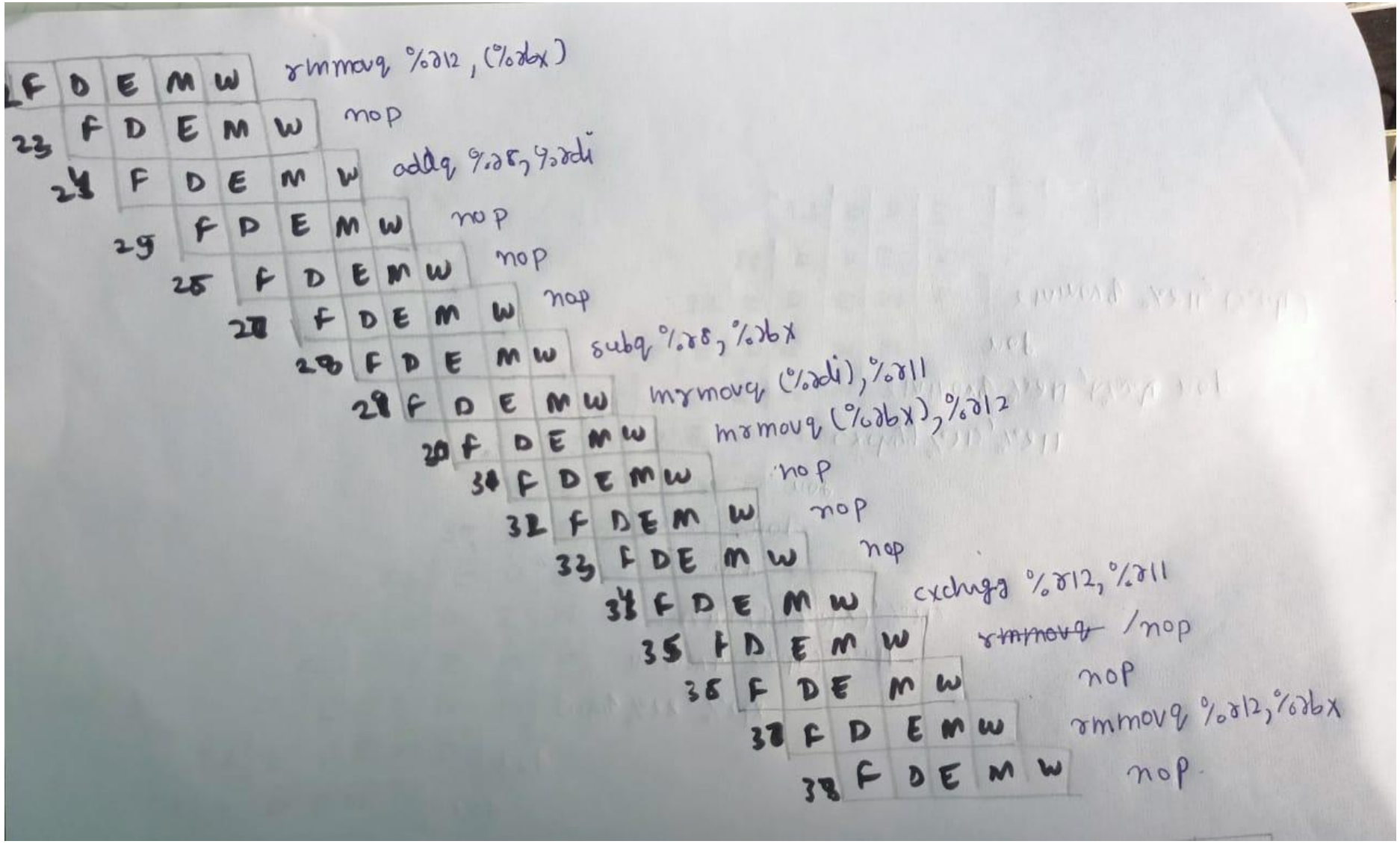
rmmovq %r12,(%rbx) (1) [as r12 was returned in second last step]
addq %r8,%rdi (1) + (1) [2 because of memory write from the previous instruction]
subq %r8,%rbx (3) + (1) [3 for structural hazard for r8 and 1 for that Cycle]

4th check or loop :
Will not have last two steps , so that will not be added for that
(1) For return

So total clock cycle = 2 + 3 + 7 + 3*(16) + 10 + 1 = 61 clock cycles

The clock cycle figure for 20 instructions which are including 37 clock cycles


F D E M W 1 irmovq stack %rsp
F D E M W 2 call main
F D E M W 3 irmovq $8, %rdi
F D E M W 4 %rsi → $8
F D E M W 5 call Loop
F D E M W 6 %r8 → $8
F D E M W 7 %r13 → $56
F D E M W 8 %rbx → %rdi
F D E M W 9 nop
F D E M W 10 nop
F D E M W 11 nop
F D E M W 12 addq %r13, %rbx
F D E M W 13
mrmovq (%rdi), %r11
F D E M W 14
mrmovq (%rbx), %r12
F D E M W 15
nop
F D E M W 16
nop
F D E M W 17
nop
F D E M W 18
cxchgq %r12, %r11
F D E M W 19
nop
F D E M W 20
nop
rmmovq
F D E M W 21
%r11, (%rbx)


rmmovq %r12, (%rbx)
nop
addq %r8, %rdi
nop
nop
nop
subq %r8, %rbx
mrmovq (%rdi), %r11
mrmovq (%rbx), %r12
nop
nop
nop
exchgq %r12, %r11
rmmovq / nop
nop
rmmovq %r12, %rbx
nop

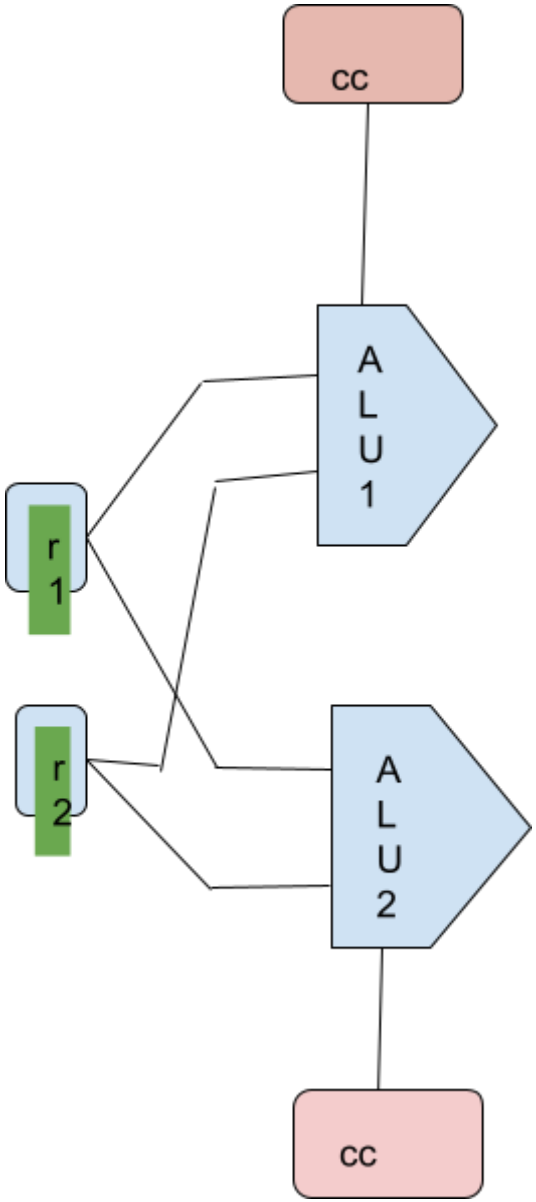
TASK 7:

Come up with any new Y86 Instruction, that will require you to add new hardware block/control signals to the Y86 architecture. Briefly describe the instruction and how it requires a new hardware block.

I am thinking of coming up with a new instruction which takes two registers Ra and Rb , and gives their division and remainder . It will disturb the original hardware by adding two ALU in the hardware.
One ALU will calculate the division of Ra and Rb and another will calculate the remainder of Ra and Rb.
1 ALU-->Ra/Rb
2 ALU-->Ra%Rb
So in the execution stage where one ALU is there, instead of it there will be two ALUs .
The condition codes also will be set twice if there will be any zero flag by dividing or by taking modulo.

We can name the instruction set as **rrdivmod ra rb**

So the new instruction also takes two registers; it will have same structure as OP instruction for add, sub etc.


cc
ALU1
r1
r2
ALU2
cc