Table of Contents

Part A) ISA intro – a recap of your ISA design

Introduction

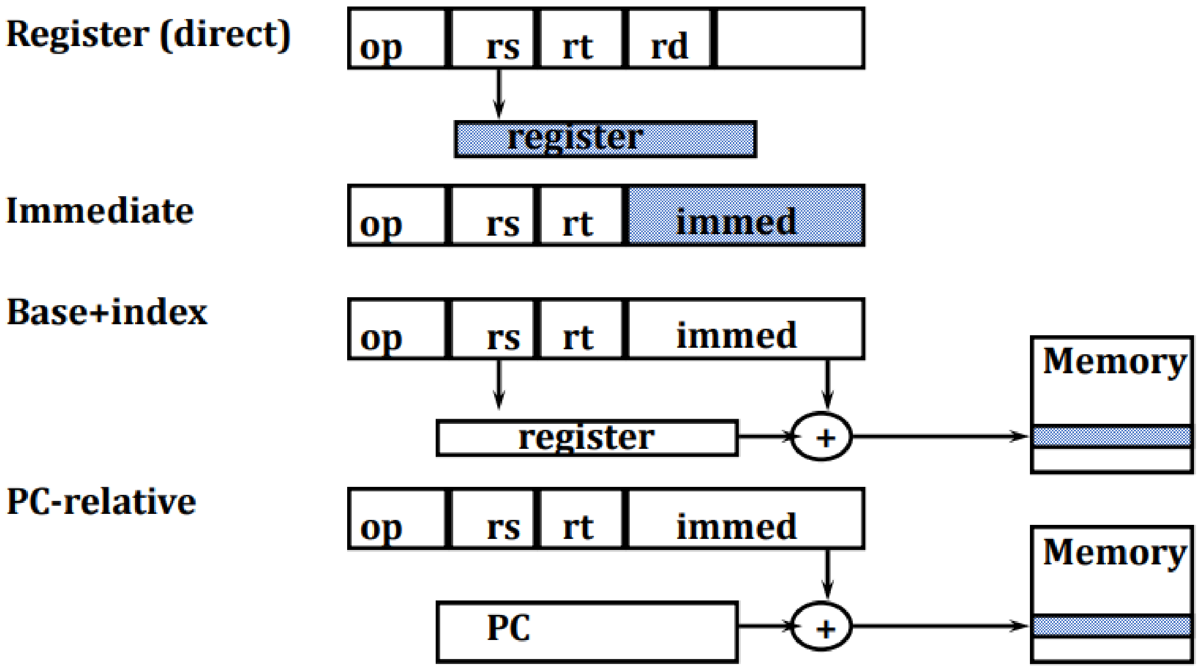
Name of the architecture: JV which is just the first letters of both our names: Joydeep Singh and Vaishnavi Medikundam
Overall Philosophy: Have an efficient ISA Design for two programs and python simulator that should work for both programs.
Specific goals strived for and achieved: Modular Exponentiation and Best match score and count

Instruction list

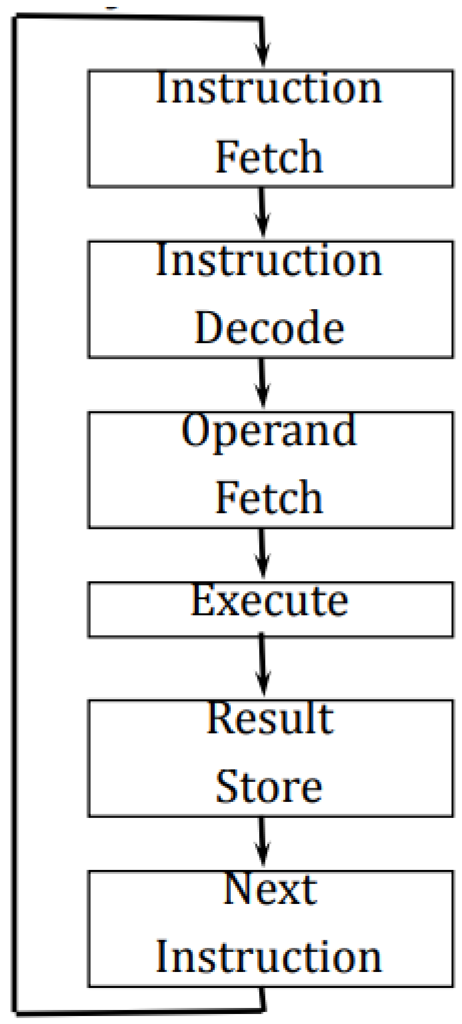
Instruction	**Functionality**	**Opcode**	**Parity Bit**
loadi *imm*	$R4 = Mem[imm]	011iiii	0
load Rx	Rx = $R4	01110 xx	1
blt4 *Rx*	$R5 = $R4 (<) Rx	00001 xx	0
beq4 *Rx*	$R5 = $R4 (==) Rx	10000 xx	1
store *Rx, Ry*	Mem[Ry] = Rx	000xx yy	1
srlRx	Rx shift right one bit, 0 shifted into MSB	001 00 xx	0
addi Rx, *imm*	Rx = Rx + imm	100 xx ii	0
subi Rx	Rx = Rx + (-imm)	101 10 xx	0
bne Rx	$R5 = Rx (!=) 0	110 11 xx	1
slt Rx	$R5 = Rx (<) 0	100 01 xx	1
BezDec *imm*	If $R5 == 0, then PC = PC + imm, else $R5 = $R5 – 1, PC = PC + 1	0100 iii	0
xor *Rx, Ry*	$R5 = Rx (EXCL) with Ry	110 xx yy	0
andi*Rx, imm*	$R5 = Rx (AND) with imm	111 xx ii	0
andi *imm*	$R5 = $R5 (AND) with imm	11110 ii	1
srl5 *imm*	$R5 shift right *imm* bits, 0 shifted into MSBs	01111 ii	1
jump *'branch'*	PC = PC -imm	010 iiii	1
add Rx, Ry	Rx = Rx+Ry	001 xx yy	1
sub Rx, Ry	Rx = Rx–Ry	101 xx yy	1
subIn $R4	$R4 = $R4 - 1	0110110	1
bne $R4	$R5 = $R4 (!=) 0	1110000	1
jump 'first branch'	PC = 12	1010101	0

halt	Stop	000 00 00	0

Register design



Control flow



Data memory addressing modes:

The addressing modes supported for data memory are:

Instruction	**Functionality**	**Opcode**	**Parity Bit**
loadi *imm*	$R4 = Mem[imm]	011 iiii	0
load Rx	Rx = $R4	01110 xx	1
store *Rx, Ry*	Mem[Ry] = Rx	000 xx yy	1

The addresses are calculated manually for each instruction and are converted to bits from MIPS assembly.

Examples of assembly load / store instructions and their corresponding machine code:

● loadi 0(0x2000)	011 0000
● store $t1, 0(0x2004)	000 0100

Part B) Answers to questions

1. The main advantage of my ISA design is the efficiency with which it has been designed. If we see instructions sets then it can be analyzed that each option is talked in this design. The main strength of the design is the register design. It can be summarized as
   I. This design is Unambiguous.
   II. The design is expressive enough. The design algorithms are expressed in the simple mapper for easy understanding
   III. Relatively have easy compilation process
   IV. The cost and performance is good for this design.

2. A “contract” amongst software and HW which Inspires compatibility, permits software and hardware to change self-sufficiently.

Functional meaning of hardware storing locations & processes.

   a. Storage sites: memory, registers
   b. Operations: multiply, add, load, branch, store, etc.
   c. Detailed explanation of in what way to raise & access them.
   d. Instructions (hardware has bit-patterns interpretations in form of the commands)

3. My learning can be summarized as follows:
   a. ISA can be defined as the functional contract.
   b. All the ISA design can be basically classified as of same type, but it all depends on in how much detail we take our design specifications to. RISC/CISC are main part in this.
   c. The matric for measurement of quality is performance, higher is the performance, good is the design.
   d. Another matric is the way we achieve compatibility between software and hardware integration. The term like binary translations etc. comes handy here.

Part C) Simulation results

1.

Pattern A

P1 DIC: 328 P2 DIC: 4739

0000000000001001

0000000000010001

0000000000000000

0000000000000000

0000000000000000

0000000000000000

Pattern B

P1 DIC: 8736 P2 DIC: 4727

0000000100001011

0001000000000011

0000000000000000

0101010101010101

0000000000000000

0000000000000000

Pattern C

P1 DIC: 4725 P2 DIC: 4729

0000000000000010

0000000000000011

0000000000000101

0000000000011101

0000000000011111

0000000000100101

Pattern D

P1 DIC: 332 P2 DIC: 4727

0000000000000000

0000000000000000

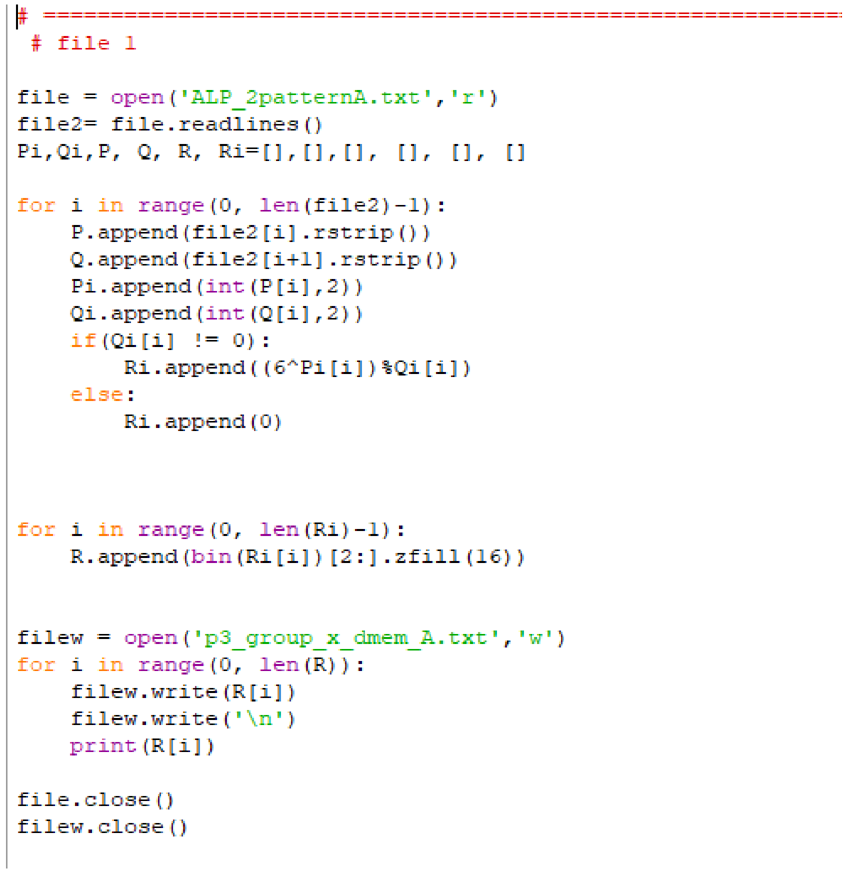
0000000000000000

1111000010001010

1111000010001011

1111000010001100

2. Execution process of the two target programs:



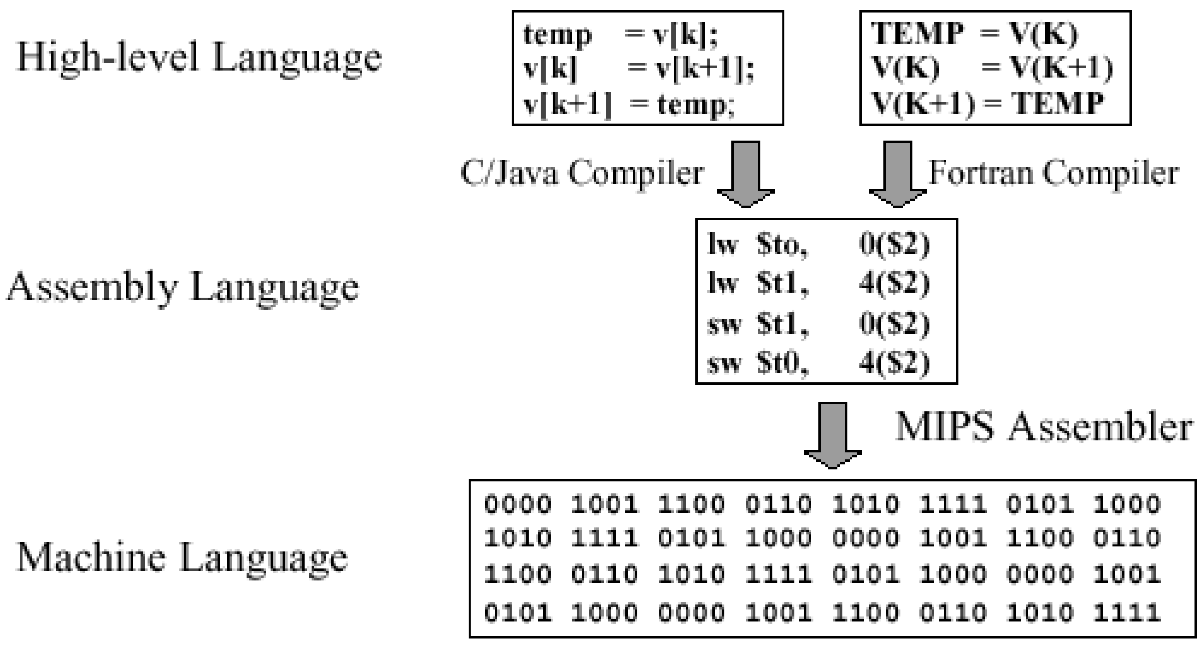
```
# ==============================================================
 # file 1

file = open('ALP_2patternA.txt','r')
file2= file.readlines()
Pi,Qi,P, Q, R, Ri=[],[],[], [], [], []

for i in range(0, len(file2)-1):
    P.append(file2[i].rstrip())
    Q.append(file2[i+1].rstrip())
    Pi.append(int(P[i],2))
    Qi.append(int(Q[i],2))
    if(Qi[i] != 0):
        Ri.append((6^Pi[i])%Qi[i])
    else:
        Ri.append(0)



for i in range(0, len(Ri)-1):
    R.append(bin(Ri[i])[2:].zfill(16))


filew = open('p3_group_x_dmem_A.txt','w')
for i in range(0, len(R)):
    filew.write(R[i])
    filew.write('\n')
    print(R[i])

file.close()
filew.close()
```

```
# # file 2

file = open('ALP_3patternB.txt','r')
file2= file.readlines()
Pi,Qi,P, Q, R, Ri=[],[],[], [], [], []

for i in range(0, len(file2)-1):
    P.append(file2[i].rstrip())
    Q.append(file2[i+1].rstrip())
    Pi.append(int(P[i],2))
    Qi.append(int(Q[i],2))
    if(Qi[i] != 0):
        Ri.append((6^Pi[i])%Qi[i])
    else:
        Ri.append(0)



for i in range(0, len(Ri)-1):
    R.append(bin(Ri[i])[2:].zfill(16))


filew = open('p3_group_x_dmem_B.txt','w')
for i in range(0, len(R)):
    filew.write(R[i])
    filew.write('\n')
    print(R[i])

file.close()
filew.close()

"
#
# # file 3
#
file = open('ALP_4patternC.txt','r')
file2= file.readlines()
Pi,Qi,P, Q, R, Ri=[],[],[], [], [], []

for i in range(0, len(file2)-1):
    P.append(file2[i].rstrip())
    Q.append(file2[i+1].rstrip())
    Pi.append(int(P[i],2))
    Qi.append(int(Q[i],2))
    if(Qi[i] != 0):
        Ri.append((6^Pi[i])%Qi[i])
    else:
        Ri.append(0)



for i in range(0, len(Ri)-1):
    R.append(bin(Ri[i])[2:].zfill(16))


filew = open('p3_group_x_dmem_C.txt','w')
for i in range(0, len(R)):
    filew.write(R[i])
    filew.write('\n')
    print(R[i])

file.close()
filew.close()
```

```
# # file 4
#
file = open('ALP_5patternD.txt','r')
file2= file.readlines()
Pi,Qi,P, Q, R, Ri=[],[],[], [], [], []

for i in range(0, len(file2)-1):
    P.append(file2[i].rstrip())
    Q.append(file2[i+1].rstrip())
    Pi.append(int(P[i],2))
    Qi.append(int(Q[i],2))
    if(Qi[i] != 0):
        Ri.append((6^Pi[i])%Qi[i])
    else:
        Ri.append(0)



for i in range(0, len(Ri)-1):
    R.append(bin(Ri[i])[2:].zfill(16))


filew = open('p3_group_x_dmem_D.txt','w')
for i in range(0, len(R)):
    filew.write(R[i])
    filew.write('\n')
    print(R[i])

file.close()
filew.close()

"
```

```
setval1 = set()                     # A new empty set
setval1.add("00")                  # Add a single member
setval1.update(["11", "10"])
setval1 |= set(["10", "11"])
if "cat" in setval1:                # Membership test
  setval1.remove("00")
setval1.discard("101")
print(setval1)
for item in setval1:                # Iteration AKA for each element
  print(item)
print("Item count:", len(setval1))
#1stitem = setval1[0]
isempty = len(setval1) == 0
setval1 = {"00011", "1111"}
#setval1 = {}
setval1 = set(["01010", "00011"])
setval2 = set(["10101", "00011"])
setval3 = setval1 & setval2             # Intersection
setval4 = setval1 | setval2             # Union
setval5 = setval1 - setval3             # Set difference
setval6 = setval1 ^ setval2             # Symmetric difference
issubset = setval1 <= setval2        # Subset test
issuperset = setval1 >= setval2      # Superset test
setval7 = setval1.copy()             # A shallow copy
setval7.remove("00011")
print(setval7.pop())
setval8 = setval1.copy()
setval8.clear()
setval9 = {x for x in range(10) if x % 2} # Set comprehension; since Python 2.7
#print(setval1, setval2, setval3, setval4, setval5, setval6, setval7, setval8, setval9, issubset, issuperset)
filew = open('p3_group_x_p1_imem.txt','w')
s1=str(setval1).split('{')
s1=s1[1].split('}')
s1=s1[0].split(',')
s2=str(setval2).split('{')
s2=s2[1].split('}')
s2=s2[0].split(',')

filew.write(s1[0])
filew.write(s1[1])
filew.close()
filew = open('p3_group_x_p2_imem.txt','w')

filew.write(s2[0])
filew.write(s2[1])

filew.close()
```

Machine Code



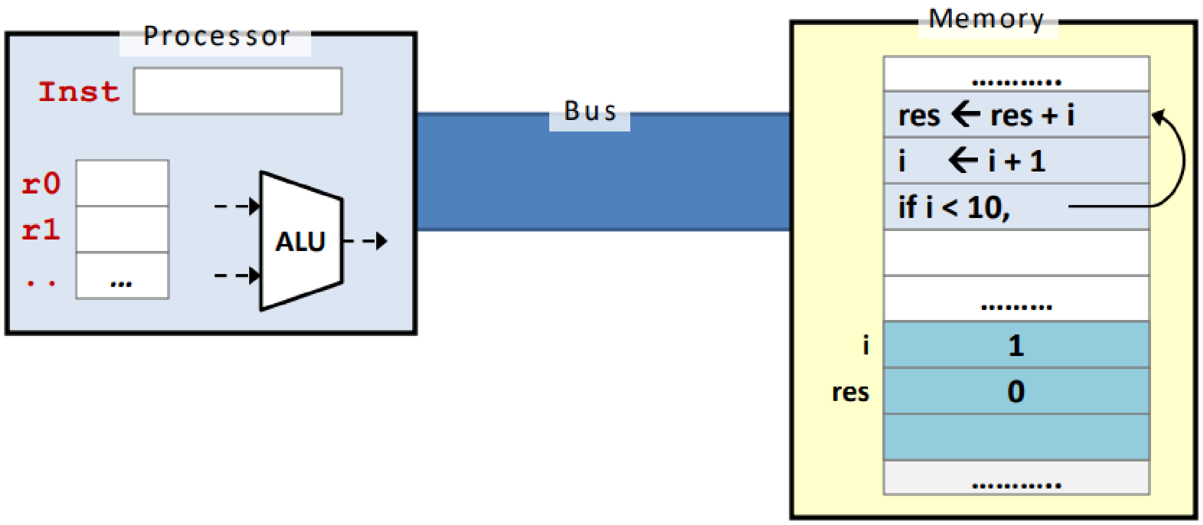
```
import dis


def machinecode(alist):
  a=10
  b=20


  if (a+b) % 2 ==0:
    print('Sum is even')
    s=a+b


  if (a+b) % 2 !=0:
    print('Sum is even')
```

```
      s=a+b-1

   return s

## Codes used

dis.dis(machinecode)
byte_code = dis.Bytecode(machinecode)
for instr in byte_code:
   print(instr.opname)

4          0 LOAD_CONST               1 (10)
           2 STORE_FAST               1 (a)

 5         4 LOAD_CONST               2 (20)
           6 STORE_FAST               2 (b)

 7         8 LOAD_FAST                1 (a)
          10 LOAD_FAST                2 (b)
          12 BINARY_ADD
          14 LOAD_CONST               3 (2)
          16 BINARY_MODULO
          18 LOAD_CONST               4 (0)
          20 COMPARE_OP               2 (==)
          22 POP_JUMP_IF_FALSE       40

 8        24 LOAD_GLOBAL              0 (print)
          26 LOAD_CONST               5 ('Sum is even')
```

```
             28 CALL_FUNCTION            1
             30 POP_TOP

  9          32 LOAD_FAST                1 (a)
             34 LOAD_FAST                2 (b)
             36 BINARY_ADD
             38 STORE_FAST               3 (s)

 11     >>   40 LOAD_FAST                1 (a)
             42 LOAD_FAST                2 (b)
             44 BINARY_ADD
             46 LOAD_CONST               3 (2)
             48 BINARY_MODULO
             50 LOAD_CONST               4 (0)
             52 COMPARE_OP               3 (!=)
             54 POP_JUMP_IF_FALSE       76

 12          56 LOAD_GLOBAL              0 (print)
             58 LOAD_CONST               5 ('Sum is even')
             60 CALL_FUNCTION            1
             62 POP_TOP

 13          64 LOAD_FAST                1 (a)
             66 LOAD_FAST                2 (b)
             68 BINARY_ADD
             70 LOAD_CONST               6 (1)
             72 BINARY_SUBTRACT
             74 STORE_FAST               3 (s)
```

```
15    >>   76 LOAD_FAST              3 (s)

           78 RETURN_VALUE
```

Part D) ISA package



1. and 2. Machine code

Algorithm and Machine Code for Program 1:

```
#Assume everything is equal to zero at first
#$t0 = 00
#$t1 = 01
#$t2 = 10
#$t3 = 11

addi $t0, 1                    0 100 00 01
addi $t1, 1                    0 100 01 01
loadi 0(0x2000)                0 011 0000
```

```
addi $t2, 3                     0 100 10 11
addi $t2, 2                     0 100 10 10
addi $t3, 3                     0 100 11 11
addi $t3, 3                     0 100 11 11
addi $t3, 3                     0 100 11 11
addi $t3, 3                     0 100 11 11
addi $t3, 3                     0 100 11 11
addi $t3, 2                     0 100 11 10

loop:
bne $s0                         1 1110000
BezDec 7                        0 0100 111

next:
bne $t2                         1 110 11 10
BezDec 4                        0 0100 100
add $t1, $t0                    1 001 01 00
subi $t2                        0 101 10 10
jump ‘next’                     0 010 1000

next2:
sub $t3, $t1                    1 101 11 01
bne $s0                         1 1110000
BezDec 7                        0 0100 111
slt $t3                         1 100 01 11
add $t3, $t1                    1 001 11 01
BezDec 3                        0 0100 011
sub $t1, $t3                    1 101 01 11
jump ‘next2’                    0 010 0111

down:
addi $t2, 3                     0 100 10 11
addi $t2, 2                     0 100 10 10
bne $s0                         1 1110000
BezDec 5                        0 0100 101
subIn $s0, 1                    1 0110110
sub $t0, $t0                    1 101 00 00
add $t0, $t1                    1 001 00 01
jump ‘loop’                     0 1010101

exit:
store $t1, 0(0x2004)            1 000 0100
halt                            0 000 0000
```

Algorithm and Machine Code for Program 2:

```
#Assume everything is equal to zero at first
#$t0 = 00
#$t1 = 01
#$t2 = 10
#$t3 = 11

addi $t0, 3                     0 100 00 11
addi $t0, 3                     0 100 00 11
addi $t0, 3                     0 100 00 11
addi $t0, 3                     0 100 00 11
addi $t0, 3                     0 100 00 11
addi $t0, 3                     0 100 00 11
addi $t0, 2                     0 100 00 10
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 3                     0 100 01 11
addi $t1, 2                     0 100 01 10

next:
loadi 0(0x200C)                 0 011 1100

next2:
loadi 0(0x2000)                 0 011 0000
load $t2                        1 01110 10
subi $t1                        0 101 10 01
xor $t2, $t3                    0 110 10 11
sub $t0, $0                     1 101 00 00
```

```
bne $t0                         1 110 11 00
BezDec 7                        0 0100 111

next3:
bne $t1                         1 110 11 01
BezDec 5                        0 0100 101
subi $t1                        0 101 10 01
andi5 1                        1 11110 01
srl5 1                          1 01111 01
jump 'next3'                    1 010 0101

next4:
bne $t0                         1 110 11 00
BezDec 7                        0 0100 111
loadi 0(0x2010)                 0 011 0000
load $t3                        1 01110 11
subi $t0                        0 101 10 00
loadi 0(0x2004)                 0 011 0100
blt4 $t1                        0 00001 01
bne $t1                         1 110 11 01
bne $t0                         1 110 11 00
BezDec 7                        0 0100 111
BezDec 5                        0 0100 101
beq4 $t1                        1 1000 01
bne $t1                         1 110 11 01
BezDec 7                        0 0100 111
jump 'first branch'             0 1010101

score:
sub $t2, $t2                    1 101 10 10
bne $t0                         1 110 11 00
BezDec 7                        0 0100 111
addi $t2, 1                     0 100 01 01
store $t1, 0(0x2004)            1 000 0100
bne $t1                         1 110 11 01
BezDec 3                        0 0100 011
store $t2, 0(0x2008)            1 000 1000
jump 'first branch'             0 1010101

best count:
sub $t2, $t2                    1 101 10 10
addi $t2, 1                     0 100 01 01
bne $t0                         1 110 11 00
```

```
BezDec 3                        0 0100 011
store $t2, 0(0x2008)            1 000 1000
jump ‘first branch’             0 1010101

exit:
halt                            0 000 0000
```

3, 4 (DATA MEM)

Pattern and Phython code

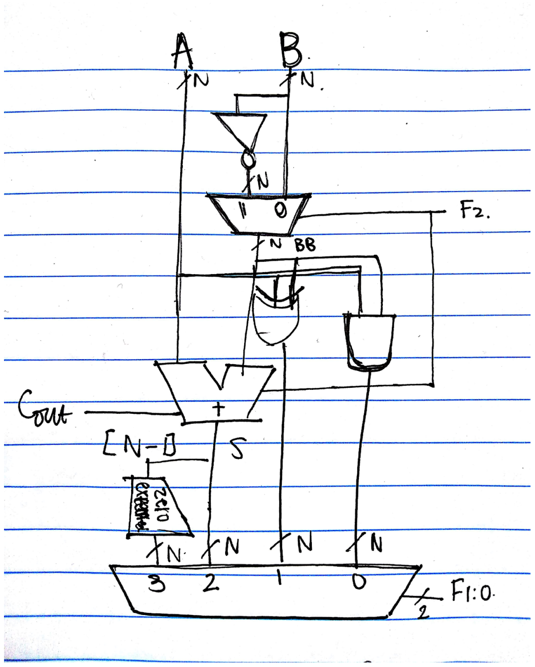
#Assume everything is equal to zero at first

```
setval1 = set()                 # A new empty set

setval1.add("00")               # Add a single member

setval1.update(["11", "10"])

setval1 |= set(["10", "11"])

if "cat" in setval1:            # Membership test

  setval1.remove("00")

setval1.discard("101")

print(setval1)

for item in setval1:            # Iteration AKA for each element

  print(item)

print("Item count:", len(setval1))

#1stitem = setval1[0]

isempty = len(setval1) == 0

setval1 = {"00011", "1111"}

#setval1 = {}

setval1 = set(["01010", "00011"])
```

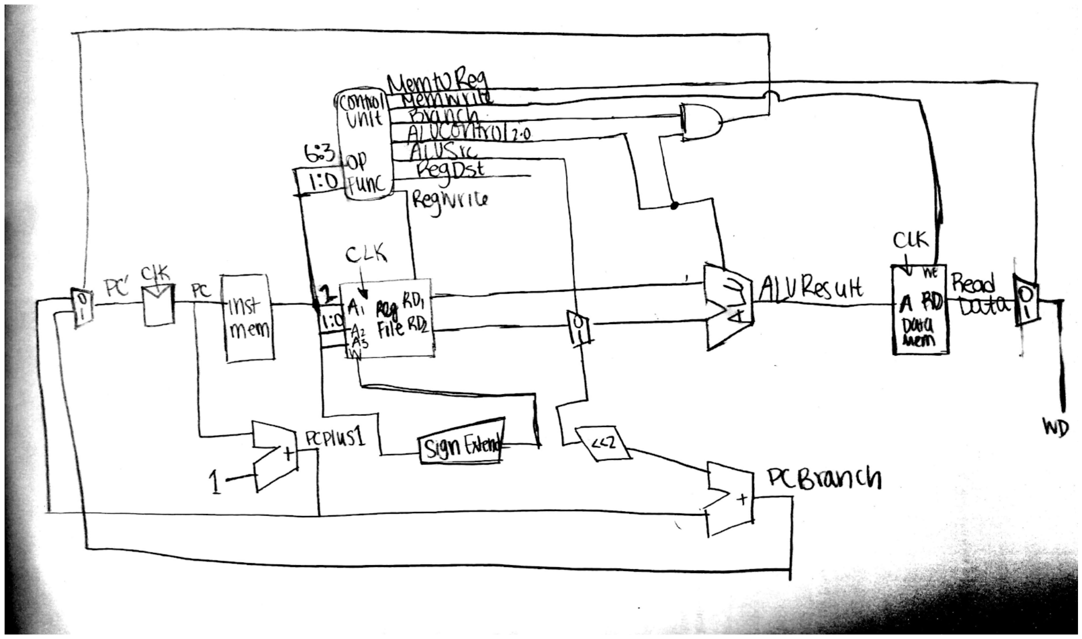
```
setval2 = set(["10101", "00011"])

setval3 = setval1 & setval2           # Intersection

setval4 = setval1 | setval2           # Union

setval5 = setval1 - setval3           # Set difference

setval6 = setval1 ^ setval2           # Symmetric difference

issubset = setval1 <= setval2       # Subset test

issuperset = setval1 >= setval2      # Superset test

setval7 = setval1.copy()            # A shallow copy

setval7.remove("00011")

print(setval7.pop())

setval8 = setval1.copy()

setval8.clear()

setval9 = {x for x in range(10) if x % 2} # Set comprehension; since Python 2.7

#print(setval1, setval2, setval3, setval4, setval5, setval6, setval7, setval8, setval9, issubset, issuperset)

filew = open('p3_group_x_p1_imem.txt','w')

s1=str(setval1).split('{')

s1=s1[1].split('}')

s1=s1[0].split(',')

s2=str(setval2).split('{')

s2=s2[1].split('}')

s2=s2[0].split(',')


filew.write(s1[0])

filew.write(s1[1])

filew.close()

filew = open('p3_group_x_p2_imem.txt','w')


filew.write(s2[0])
filew.write(s2[1])


filew.close()
```

5. HW Schematics

A. ALU Schematic



B. CPU Datapath design



C. Control logic design

INST	OP	RegWrt	ALUctr	Branch	Memwrt	MemtoReg
Load	1 000	1	1	0	1	1
Store	1 011	0	1	0	0	0
shl	0 001	1	0	0	0	0
sll	0 100	1	0	0	0	0
BezDec	0 010	0	0	1	0	0
xori	0 110	1	0	0	0	0
andi	0 111	1	0	0	0	0
jump	1 010	0	0	1	0	1
sub	1 101	1	0	0	0	0
loadi	0 011	1	1	0	1	1
blt4	0 000	0	0	0	0	0
beq4	1 100	0	0	0	0	0
andi5	1 111	1	0	0	0	0
srl5	1 011	1	0	0	0	0