Table of Contents

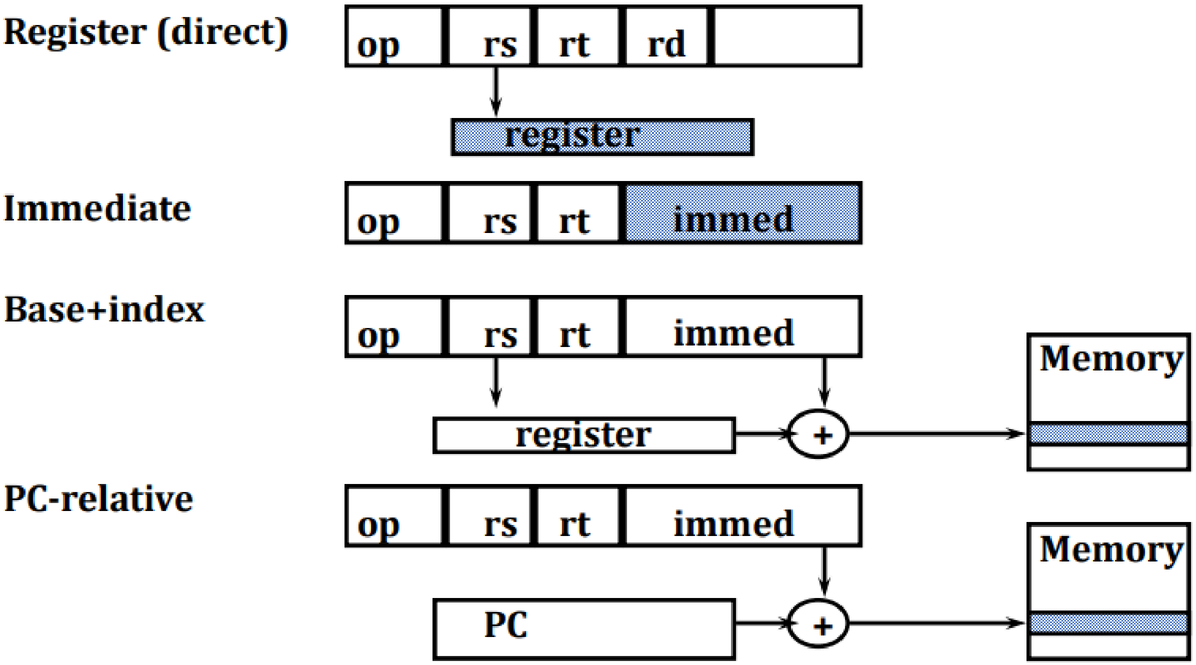
Part A) ISA intro – a recap of your ISA design

Introduction

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
andi5*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	101xx yy	1
subln $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



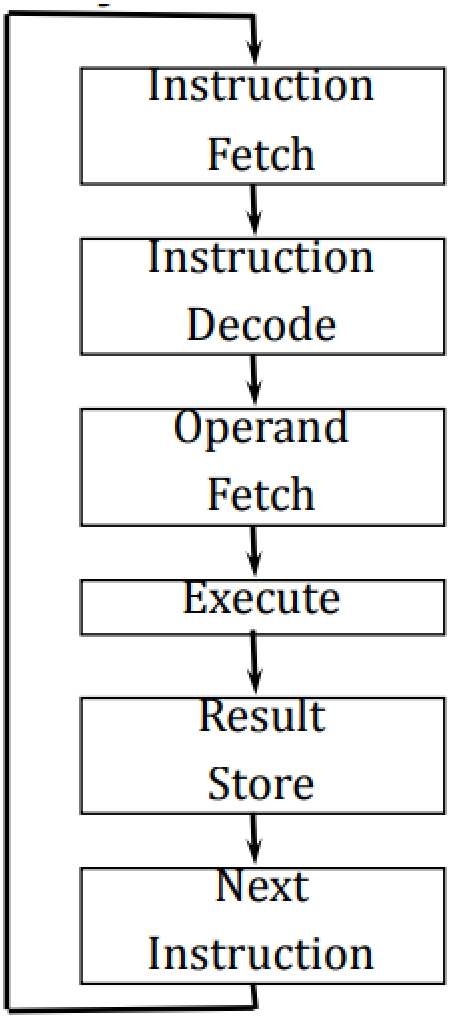
Register (direct)

Immediate

Base+index

PC-relative

Control flow



Data memory addressing modes

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. The program for applying formula based on P and Q

```
# 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()
```

2.

```
# file 3

file = open('ALP_4patternC.txt','r')
```

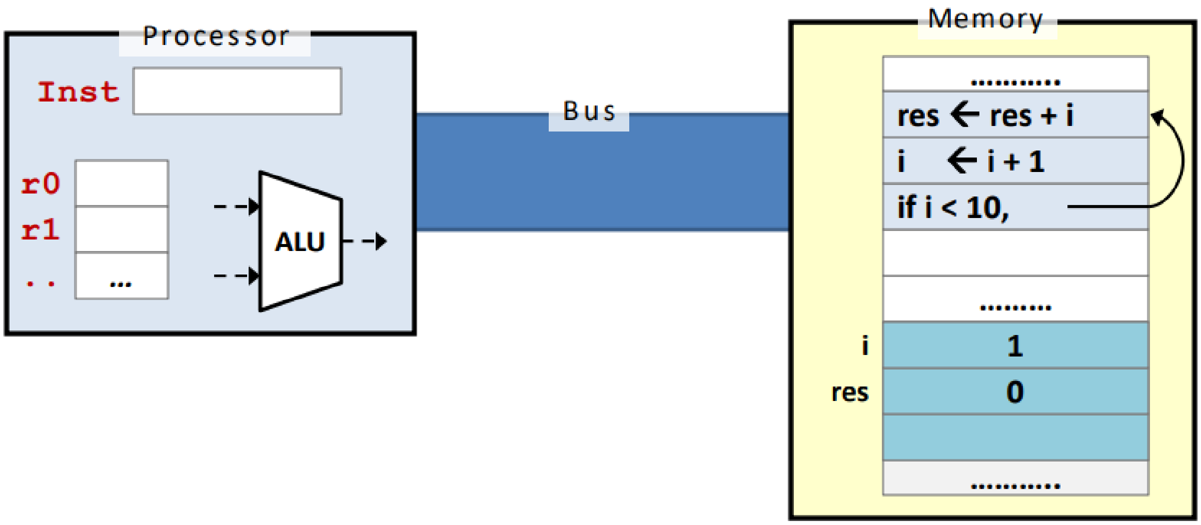
```
file2= file.readlines()
Pi,Qi,P, Q, R, Ri=[],[],[], [], [], []

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()
```

Part D) ISA package

1. Algorithm



2. Machine code

Machine Code for Program 1:

```
#Assume everything is equal to zero at first

#$t0 = 00

#$t1 = 01

#$t2 = 10

#$t3 = 11
```

```
addi $t0, 1                              100 00 01
addi $t1, 1                              100 01 01
loadi 0(0x2000)                  0110000
addi $t2,3                               100 10 11
addi $t2, 2                              100 10 10
addi$t3, 3                               100 11 11
addi $t3, 3                              100 11 11
addi $t3, 3                              100 11 11
addi $t3, 3                              100 11 11
addi $t3, 3                              100 11 11
addi $t3, 2                              100 11 10

loop:
bne$s0                           1110000
BezDec7                                  0100 111

next:
bne $t2                          110 11 10
BezDec4                                  0100 100
add $t1, $t0                             001 01 00
subi $t2                         101 10 10
jump 'next'                              010 1000

next2:
sub $t3, $t1                             101 11 01
bne $s0                          1110000
BezDec7                                  0100 111
slt $t3                                  100 01 11
```

```
add $t3, $t1                001 11 01
BezDec 3                    0100 011
sub $t1, $t3                101 01 11
jump 'next2'                010 0111

down:
addi $t2, 3                 100 10 11
addi $t2, 2                 100 10 10
bne $s0               1110000
BezDec5                     0100 101
subIn$s0, 1                 0110110
sub $t0, $t0                101 00 00
add $t0, $t1                001 00 01
jump 'loop'                 1010101

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

Machine Code for Program 2:

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

addi $t0,3                  100 00 11
addi $t0,3                  100 00 11
addi $t0,3                  100 00 11
addi $t0,3                            100 00 11

addi $t0,3                            100 00 11

addi $t0,3                            100 00 11

addi $t0,2                            100 00 10

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 3                           100 01 11

addi $t1, 2                           100 01 10


next:

loadi 0(0x200C)               011 1100


next2:

loadi 0(0x2000)               011 0000

load $t2                              01110 10

subi$t1                       101 10 01

xor $t2, $t3                          110 10 11

sub $t0, $0                           101 00 00

bne $t0                               110 11 00

BezDec7                               0100 111


next3:
```

```
bne $t1                                 110 11 01
BezDec5                                           0100 101
subi $t1                                101 10 01
andi5 1                                           11110 01
srl5 1                                            01111 01
jump ‘next3’                                      010 0101

next4:
bne $t0                                           110 11 00
BezDec7                                           0100 111
loadi 0(0x2010)                         011 0000
load $t3                                          01110 11
subi$t0                                 101 10 00
loadi 0(0x2004)                         011 0100
blt4 $t1                                0000101
bne $t1                                 110 11 01
bne $t0                                           110 11 00
BezDec7                                           0100 111
BezDec5                                           0100 101
beq4 $t1                                          1000 01
bne $t1                                 110 11 01
BezDec7                                           0100 111
jump ‘first branch’                               1010101

score:
sub $t2, $t2                                      101 10 10
bne $t0                                           110 11 00
BezDec7                                           0100 111
addi $t2, 1                                       100 01 01
```

```
store $t1, 0(0x2004)                    000 0100
bne $t1                         110 11 01
BezDec3                                 0100 011
store $t2, 0(0x2008)                    000 1000
jump ‘first branch’                     1010101

best count:
sub $t2, $t2                            101 10 10
addi $t2, 1                             100 01 01
bne $t0                                 110 11 00
BezDec3                                 0100 011
store $t2, 0(0x2008)                    000 1000
jump ‘first branch’                     1010101

exit:
halt                                    000 0000
```

3, 4

Pattern and 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()
```