COMPUTER SYSTEMS ORGANISATION

NAME-HARSHITA GUPTA
ROLL NUMBER-2020101078

ASSIGNMENT 2
ARCHITECTURAL EXPLORATION

TASK 1

Instruction Encoding

Byte	0		1	
cxchng[XX] rA , rB	C	XX	rA	rB

Condition [XX]

byte	0	
exchngnc	C	0
exchngle	C	1
exchngl	C	2
exchnge	C	3
exchngne	C	4
exchngge	C	5
exchangg	C	7

Our new instruction “exchng[XX] rA, rB” will be 2 bytes long.

The first byte is the “instruction specifier” byte which has 2 parts (4 bits each), “iCode and iFun”, iCode specifies that the instruction is of type exchng and iFun specifies the condition for exchange.

[XX] can be anyone of { ‘nc’, ‘le’, ‘l’, ’e’, ‘ne’, ‘ge’, ‘g’ }, which include unconditional exchange and conditional exchange i.e for example exchngl rA, rB, the value in rA and rB will be swapped/exchanged if and only if rB is less than rA.

The second byte will be the “register specifier” byte which again has two parts (4 bits each) specifying the register ID of rA and rB respectively.

TASK 2

```
      .pos 0
      irmovq stack, %rsp    # Set up stack pointer
      jmp main              # Execute main program
                            # Array of 8 elements of size 8 bytes each
      .align 8
array:
      .quad 0x7
      .quad 0x5
      .quad 0x1
      .quad 0x3
      .quad 0x4
      .quad 0x0
      .quad 0x6
      .quad 0x2
main:
      irmovq array, %rdi          # Base address of the array
      rrmovq %rdi, %rsi           # %rsi = %rdi = &array[0]
      irmovq $56, %r8
      addq   %r8, %rsi            # Add 56 to %rsi, %rsi = &array[7]
      irmovq $0, %r8              # Iterator i
      irmovq $1, %r12
      irmovq $4, %r13
      irmovq $8, %r14
loop:
      mrmovq (%rdi), %r10         # Get the value of array[i] in %r10
      mrmovq (%rsi), %r11         # Get the value of array[7-i] in %r11
      rrmovq   %r11, %rcx
      subq    %r10, %rcx          # Set the Condition Codes for the exchng

      exchngg %r10, %r11          # if %r11 > %r10 i.e. if arr[7-i] > arr[i]
                                  # then swap the values in the registers
      rmmovq %r10, (%rdi)
      rmmovq %r11, (%rsi)         # Writeback the values in memory
      jmp check
check:
      addq %r12, %r8              # increment i
      subq %r8, %r13              # Set CC for i == 4
      irmovq $4, %r13
      je   done
      addq %r14, %rdi       # point %rdi to the next element from the beginning
      subq %r14, %rsi       # point %rsi to the previous element from the end
      jmp loop
done:
      halt

      .pos 0x200
stack:
```

TASK 3

0x0000:		.pos 0
0x0000:	30f40000200000000000	irmovq stack, %rsp
0x000a:	705800000000000000	jmp main
0x0013:		.align 8
0x0018: 0x0018: 0x0020: 0x0028: 0x0030: 0x0038: 0x0040: 0x0048: 0x0050:	 07 05 01 03 04 00 06 02	array: .quad 0x7 .quad 0x5 .quad 0x1 .quad 0x3 .quad 0x4 .quad 0x0 .quad 0x6 .quad 0x2
0x0058:		main:
0x0058:	30f71800000000000000	irmovq array, %rdi
0x0062:	2076	rrmovq %rdi, %rsi
0x0064:	30f83800000000000000	irmovq $56, %r8
0x006e:	6086	addq %r8, %rsi
0x0070:	30f80000000000000000	irmovq $0, %r8
0x007a:	30fc0100000000000000	irmovq $1, %r12
0x0084:	30fd0400000000000000	irmovq $4, %r13
0x008e:	30fe0800000000000000	irmovq $8, %r14
0x0098:		loop:
0x0098:	50a70000000000000000	mrmovq (%rdi), %r10
0x00a2:	50b60000000000000000	mrmovq (%rsi), %r11
0x00ac:	20b1	rrmovq %r11, %rcx
0x00ae:	61a1	subq %r10, %rcx
0x00b0:	c5ab	exchngg %r10, %r11
0x00b2:	40a70000000000000000	rmmovq %r10, (%rdi)
0x00bc:	40b60000000000000000	rmmovq %r11, (%rsi)
0x00c6:	70cf0000000000000000	jmp check
0x00cf:		check:
0x00cf:	60c8	addq %r12, %r8
0x00d1:	618d	subq %r8, %r13
0x00d3	30fd0400000000000000	irmovq $4, %r13
0x00dd:	73f300000000000000	je done
0x00e6:	60e7	addq %r14, %rdi
0x00e8:	61e6	subq %r14, %rsi

0x00ea:	709800000000000000	jmp loop
0x00f3:		done:
0x00f3:	00	halt
0x00f4:		.pos 0x200
0x0200:		stack:

TASK 4

Stage	exchng[XX] rA, rB
Fetch	iCode:iFun <- M_1[PC] rA:rB <- M_1[PC + 1] valP <- PC + 2
Decode	valA <- R[rA] valB <- R[rB]
Execute	Cnd <- Cond(CC, ifun)
Memory	
Write back	R[rA] <- Cnd ? valB : valA R[rB] <- Cnd ? valA : valB
PC Update	PC <- valP

TASK 5

Cycle ID	Instruction	PC	SF	ZF	OF	Changes in General Purpose Registers	Changes in Memory
1	irmovq stack, %rsp	0x000a	0	0	0	%rsp = 0x0200	
2	jmp main	0x0058	0	0	0		0x0018 0700000000000000 0x0020 0500000000000000 0x0028 0100000000000000 0x0030 0300000000000000 0x0038 0400000000000000 0x0040 0000000000000000 0x0048 0600000000000000 0x0050 0200000000000000
3	irmovq array, %rdi	0x0062	0	0	0	%rdi = 0x0018	
4	rrmovq %rdi, %rsi	0x0064	0	0	0	%rsi = 0x0018	
5	irmovq $56, %r8	0x006e	0	0	0	%r8 = 0x0038	
6	addq %r8, %rsi	0x0070	0	0	0	%rsi = 0x0050	
7	irmovq $0, %r8	0x007a	0	0	0	%r8 = 0x0000	
8	irmovq $1, %r12	0x0084	0	0	0	%r12 = 0x0001	
9	irmovq $4, %r13	0x008e	0	0	0	%r13 = 0x0004	
10	irmovq $8, %r14	0x0098	0	0	0	%r14 = 0x0008	
11	mrmovq (%rdi), %r10	0x00a2	0	0	0	%r10 = 0x0007	
12	mrmovq (%rsi), %r11	0x00ac	0	0	0	%r11 = 0x0002	
13	rrmovq %r11, %rcx	0x00ae	0	0	0	%rcx = 0x0002	
14	subq %r10, %rcx	0x00b0	1	0	0	%rcx = 0xfffffffffffffffb	

15	exchngg %r10, %r11	0x00b2	1	0	0	No Change	
16	rmmovq %r10, (%rdi)	0x00bc	1	0	0		0018 0700000000000000 0020 0500000000000000 0028 0100000000000000 0030 0300000000000000 0038 0400000000000000 0040 0000000000000000 0048 0600000000000000 0050 0200000000000000
17	rmmovq %r11, (%rsi)	0x00c6	1	0	0		0018 0700000000000000 0020 0500000000000000 0028 0100000000000000 0030 0300000000000000 0038 0400000000000000 0040 0000000000000000 0048 0600000000000000 0050 0200000000000000
18	jmp check	0x00cf	1	0	0		
19	addq %r12, %r8	0x00d1	0	0	0	%r8 = 0x0001	
20	subq %r8, %r13	0x00d3	0	0	0	%r13 = 0x0003	
21	irmovq $4, %r13	0x00dd	0	0	0	%r13 = 0x0004	
22	je done	0x00e6	0	0	0		

TASK 6

Number of cycles required by the program to execute via the pipeline implementation of Y86-64 = 82

```
        .pos 0
        Instr 1 (Completed on cycle 5) irmovq stack, %rsp
        Instr 2 (Completed on cycle 6) jmp main
        .align 8
array:
        .quad 0x7
        .quad 0x5
```

```
        .quad 0x1
        .quad 0x3
        .quad 0x4
        .quad 0x0
        .quad 0x6
        .quad 0x2
main:
        Instr 3  (Completed on cycle 7)     irmovq array, %rdi
        Instr 4  (Completed on cycle 8)     rrmovq %rdi, %rsi
        Instr 5  (Completed on cycle 9)     irmovq $56, %r8
Executed  Instr 6  (Completed on cycle 10)    addq   %r8, %rsi
only    Instr 7  (Completed on cycle 11)    irmovq $0, %r8
once    Instr 8  (Completed on cycle 12)    irmovq $1, %r12
        Instr 9  (Completed on cycle 13)    irmovq $4, %r13
        Instr 10 (Completed on cycle 14)   irmovq $8, %r14
loop:
        Instr 11 (Completed on cycle 15, 33, 51, 69)  mrmovq (%rdi), %r10
        Instr 12 (Completed on cycle 16, 34, 52, 70)  mrmovq (%rsi), %r11
        Instr 13 (Completed on cycle 18, 36, 54, 72 due to bubble)  rrmovq   %r11, %rcx
        Instr 14 (Completed on cycle 19, 37, 55, 73)  subq      %r10, %rcx
        Instr 15 (Completed on cycle 20, 38, 56, 74)  exchngg %r10, %r11
        Instr 16 (Completed on cycle 21, 39, 57, 75)  rmmovq %r10, (%rdi)
        Instr 17 (Completed on cycle 22, 40, 58, 76)  rmmovq %r11, (%rsi)
        Instr 18 (Completed on cycle 23, 41, 59, 77)   jmp check
check:
        Instr 19 (Completed on cycle 24, 42, 60, 78)  addq %r12, %r8
        Instr 20 (Completed on cycle 25, 43, 61, 79)  subq %r8, %r13
        Instr 21 (Completed on cycle 26, 44, 62, 80)  irmovq $4, %r13
        Instr 22 (Completed on cycle 27, 45, 63, 81)  je   done
        Instr 23 (Completed on cycle 30, 48, 66 2 bubbles due to mispredicted branch) addq %r14,%rdi
        Instr 24 (Completed on cycle 31, 49, 67 )  subq %r14, %rsi
        Instr 25 (Completed on cycle 32, 50, 68)  jmp loop
done:
        Instr 26 (Completed on cycle 82)   halt
        .pos 0x200
stack:
```

Instructions	Comments, Clock Cycle ->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
irmovq array, %rdi		F	D	E	M	W																				
rrmovq %rdi, %rsi	Use forwarding, e_ValE = ValA		F	D	E	M	W																			
irmovq $56, %r8				F	D	E	M	W																		
addq %r8, %rsi	Use forwarding, e_ValE = ValA				F	D	E	M	W																	
irmovq $0, %r8						F	D	E	M	W																
irmovq $1, %r12							F	D	E	M	W															
irmovq $4, %r13								F	D	E	M	W														
irmovq $8, %r14									F	D	E	M	W													
mrmovq (%rdi), %r10										F	D	E	M	W												
mrmovq (%rsi), %r11											F	D	E	M	W											
bubble	Add a bubble in the fetch stage											F														
rrmovq %r11, %rcx	Use forwarding, m_ValM = ValA												F	D	E	M	W									
subq %r10, %rcx	Use forwarding,e_ValE = ValB													F	D	E	M	W								
exchngg %r10, %r11															F	D	E	M	W							
rmmovq %r10, (%rdi)	Use forwarding e_ValE = valA															F	D	E	M	W						
rmmovq %r11, (%rsi)	Use forwarding, m_ValE = valA																F	D	E	M	W					
jmp check																		F	D	E	M	W				
addq %r12, %r8																			F	D	E	M	W			
subq %r8, %r13	Use forwarding, ValA = e_ValB																			F	D	E	M	W		
irmovq $4, %r13																					F	D	E	M	W	
je done																						F	D	E	M	W

TASK 7

Consider a instruction absDiff rA, rB which works in the following way, rB = rB > rA ? rB – rA : rA – rB.
This cannot be implemented using existing Y-86 architecture because it involves atleast two computations at the ALU level.
This can be implemented by adding another ALU in the execute stage, so ALU1 will compute ValE1 = rB – rA and set the CC1 accordingly. ALU2 will similarly compute ValE2 = rA – rB and set the CC2 accordingly. Then there will be a Compute Cnd block which will be say, equal to 1 if rB >= rA i.e rB -rA > 0 i.e. if CC1=000 or ZF1=1, SF1=0, OF1=0 OR will be

equal to 0 if rB < rA i.e rA – rB > 0 i.e. CC2 =000. Then based on this value of Cnd, rB = ValE1 or ValE2 in the write back stage.

Stage	absDiff rA, rB
Fetch	iCode:iFun <- M_1[PC] rA:rB <- M_1[PC + 1] valP <- PC + 2
Decode	valA <- R[rA] valB <- R[rB]
Execute	ValE1 = valB – valA ValE2 = valA – valB Set CC1 Set CC2 Cnd <- Cond(CC1, CC2)
Memory	
Write back	R[rB] <- Cnd ? valE1 : valE2
PC Update	PC <- valP