CSO-Assignment-3

Name: gaddam vinaychandra

Roll no: 2020101010.

Instruction:

cxchng[XX] rA,rB

Task 1:

Encoding for given instruction:

cxchng[XX] rA,rB

C	Fn	rA	rB

Here 'C' is hexadecimal number indicates that it is exchange instruction and 'Fn' specifies condition to swap or exchange. rA and rB are register specifiers.

Instruction specifiers for different conditions

Xchng	C	0
Cxchngg	C	6
Cxchngl	C	2
cxchngne	C	4

ge,le,e are not there for this instruction because swapping in 'e' condition is useless

it is trivial that instruction is of two bytes

Task 2:

code to problem 3:

```
      .pos 0                          #instructions start from address 0
       irmovq  stack,%rsp              #setup stack pointer
       call main                       #go to main
       halt                            #terminate program
       #array of 8 elements
       .align 8
array:
        .quad 0x726
        .quad 0x163
        .quad 0x2
        .quad 0x716
        .quad 0x8
        .quad 0x27
        .quad 0x4
        .quad 0x0826

main:
       irmovq array,%rdi
       irmovq $4,%rsi
       call swap                       #swap(array,4)
       ret
       #void swap(long array[],long halfsize)
       #arguments are stored in %rdi & %rsi
swap:

       irmovq $8,%r8                   #const 8
       irmovq $1,%r9                   #const 1
       rrmovq %rdi %rbx                #array pointer
       irmovq $56,%rcx                 #const 56
       addq %rcx,%rbx                  #last elem pointer in %rbx

loop:
       mrmovq (%rdi),%r10              #array[i] to %r10
       mrmovq (%rbx),%r11              #array[7-i or l] to %r11
    ** cxchngg %r10,%r11               #swap if %r11 > %r10
       rmmovq %r10,(%rdi)              #%r10 to array[i]
       rmmovq %r11,(%rbx)              #%r11 to array[7-i]
       addq %r8,%rdi                   #i=i+1
       subq %r8,%rbx                   #l=l-1
       subq %r9,%rsi                   #count--
       jne  loop                       #stop when 0
       ret                             #return
      .pos 0x200
      Stack:
```

Task 3:

```
0x000:                                .pos 0                #instructions start from address 0
0x000:30f40003000000000000            irmovq  stack,%rsp          #setup stack pointer
0x00a:805800000000000000              call main                   #go to main
0x013:00                              halt                        #terminate program

                                                                  #array of 8 elements
0x018:                                .align 8
0x018:                            array:
0x018:2607000000000000                .quad 0x726
0x020:6301000000000000                .quad 0x163
0x028:0200000000000000                .quad 0x2
0x030:1607000000000000                .quad 0x716
0x038:0800000000000000                .quad 0x8
0x040:2700000000000000                .quad 0x27
0x048:0400000000000000                .quad 0x4
0x050:2608000000000000                .quad 0x0826

0x058:                            main:

0x058:30f71800000000000000            irmovq array,%rdi
0x062:30f60400000000000000            irmovq $4,%rsi
0x06c:807600000000000000              call swap                   #swap(array,4)
0x075:90                              ret
                                      #void swap(long array[],long halfsize)
                                      #arguments are stored in %rdi & %rsi
0x076:                            swap:
0x076:30f80800000000000000            irmovq $8,%r8               #const 8
0x080:30f90100000000000000            irmovq $1,%r9               #const 1
0x08a:2073                            rrmovq %rdi %rbx            #array pointer
0x08c:30f13800000000000000            irmovq $56,%rcx             #const 56
0x096:6013                            addq %rcx,%rbx              #last elem pointer in %rbx

0x098:                            loop:
0x098:50a70000000000000000            mrmovq (%rdi),%r10          #array[i] to %r10
0x0a2:50b30000000000000000            mrmovq (%rbx),%r11          #array[7-i or l] to %r11
0x0ac:c6ab                          **cxchngg %r10,%r11           #swap if %r11 > %r10
0x0ae:40a70000000000000000            rmmovq %r10,(%rdi)          #%r10 to array[i]
0x0b8:40b30000000000000000            rmmovq %r11,(%rbx)          #%r11 to array[7-i]
0x0c2:6087                            addq %r8,%rdi               #i=i+8
0x0c4:6183                            subq %r8,%rbx               #l=l-1
0x0c6:6196                            subq %r9,%rsi               #count--
0x0c8:749800000000000000              jne  loop                   #stop when 0
0x0d1:90                              ret                         #return
0x300:                                .pos 0x300
0x300:                            stack:
```

Task 4:

Here we assume that there is control logic block to select value of valE destination port from valA,valB,valE and another control logic to select value of valM destination port from valA,valB,valM also given instruction can change CC other wise it's impossible to execute this instruction in one cycle

Below is the instruction execution stages for first instance in first iteration of loop marked with red star mark in above code

Stage	1^{st} instance	cxchngXX rA,rB
Fetch	icode=2:ifun=6 rA=a:rB=b valP=0xac+2=0xae	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC+2
Decode	valA=0x726 valB=0x826	valA ← R[rA] valB ← R[rB]
Execute	valE=-(0x100) Cnd=1	valE ← valA-valB Cnd ← cond(CC,ifun)
Memory	---	---
Write back	R[rA]=valB=0x826 R[rB]=valA=0x726	R[rA] ← Cnd? valB:valA R[rB] ← Cnd? valA:valB
PC update	PC=valP=0xae	PC ← valP

Task 5:

Below table reports the value of program counter ,condition code registers(order ZF,SF,OF),general purpose registers (*only used ones*) ,stack pointer(if used) and any modified address in memory at the end of particular cycle mentioned

GPR=general purpose registers(if not mentioned they are '0')
CC=condition codes
%rsp=stack pointer
'***'= CC are updated in particular instruction

Cycle no	instruction	GPR	CC	Memory	PC	%rsp
1	irmovq stack,%rsp	All '0'	000	---	0x0a	0x300
2	call main	All '0'	000	0x300←0x13	0x58	0x2f8
3	irmovq array,%rdi	%rdi←0x18	000	---	0x62	0x2f8
4	irmovq $4,%rsi	%rsi←4 %rdi=0x18	000	---	0x6c	0x2f8
5	call swap	%rsi=4 %rdi=0x18	000	0x2f8←0x75	0x76	0x2f0
6	irmovq $8,%r8	%r8←8 %rsi=4 %rdi=0x18	000	---	0x80	0x2f0
7	Irmovq $1,%r9	%r9←1 %r8=8 %rsi=4 %rdi=0x18	000	---	0x8a	0x2f0
8	rrmovq %rdi,%rbx	%rbx←0x18 %r9=1 %r8=8 %rsi=4 %rdi=0x18	000	---	0x8c	0x2f0
9	irmov $56,%rcx	%rcx←56 %rbx=0x18	000	---	0x96	0x2f0

		%r9=1 %r8=8 %rsi=4 %rdi=0x18				
10	addq %rcx,%rbx	%rbx←0x50 %rcx=56 %r9=1 %r8=8 %rsi=4 %rdi=0x18	000 ***	---	0x98	0x2f0
11	mrmovq(%rdi),%r10	%r10←0x726 %rbx=0x50 %rcx=56 %r9=1 %r8=8 %rsi=4 %rdi=0x18	000	---	0xa2	0x2f0
12	mrmovq(%rbx),%r11	%r11←0x826 %r10=0x726 %rbx=0x50 %rcx=56 %r9=1 %r8=8 %rsi=4 %rdi=0x18	000	---	0xac	0x2f0
13	cxchngg %r10,%r11	%r11←0x726 %r10←0x826 %rbx=0x50 %rcx=56 %r9=1 %r8=8 %rsi=4 %rdi=0x18	000 ***	---	0xae	0x2f0
14	rmmovq%r10,(%rdi)	%r11=0x726 %r10=0x826 %rbx=0x50 %rcx=56 %r9=1 %r8=8 %rsi=4	000	0x18←0x826	0xb8	0x2f0

		%rdi=0x18				
15	rmmovq%r11,(%rbx)	%r11=0x726 %r10=0x826 %rbx=0x50 %rcx=56 %r9=1 %r8=8 %rsi=4 %rdi=0x18	000	0x50←0x726	0xc2	0x2f0
16	addq %r8,%rdi	%rdi←0x20 %r11=0x726 %r10=0x826 %rbx=0x50 %rcx=56 %r9=1 %r8=8 %rsi=4	000 ***	---	0xc4	0x2f0
17	subq %r8,%rbx	%rbx←0x48 %rdi=0x20 %r11=0x726 %r10=0x826 %rcx=56 %r9=1 %r8=8 %rsi=4	000 ***	---	0xc6	0x2f0
18	Subq %r9,%rsi	%rsi←3 %rbx=0x48 %rdi=0x20 %r11=0x726 %r10=0x826 %rcx=56 %r9=1 %r8=8	000 ***	---	0xc8	0x2f0
19	jne loop	%rsi=3 %rbx=0x48 %rdi=0x20 %r11=0x726 %r10=0x826 %rcx=56 %r9=1	000	---	0x98	0x2f0

		%r8=8				
20	mrmovq(%rdi),%r10	%r10←0x163 %rsi=3 %rbx=0x48 %rdi=0x20 %r11=0x726 %rcx=56 %r9=1 %r8=8	000	---	0xa2	0x2f0

Task 6:

Finding number of cycles required to execute the above program:

Cycles required = i+4+x

Where i =total number of instructions to be executed

4=because first instruction completes execution after '5' cycles

X=any extra cycles wasted for bubble or squashing to avoid data hazards & control hazards.

Instruction execution sequence

1.irmovq 2.call main 3.irmovq 4.irmovq 5.call swap 6.irmovq 7.irmovq 8.rrmovq 9.irmovq 10.addq 11.mrmovq 12.mrmovq (1 bubble) 13.cxchngg 14.rmmovq 15.rmmovq 16.addq 17.subq 18.subq 19.jne 20.mrmovq 21.mrmovq (1 bubble) 22.cxchngg 23.rmmovq 24.rmmovq 25.addq 26.subq 27.subq 28.jne 29.mrmovq 30.mrmovq (1 bubble) 31.cxchngg 32.rmmovq 33.rmmovq 34.addq 35.subq 36.subq 37.jne 38.mrmovq 39.mrmovq (1 bubble) 40.cxchngg 41.rmmovq 42.rmmovq 43.addq 44.subq 45.subq 46.jne (2 cycles wasted for wrong prediction) 47.ret (3 cycles wasted) 48.ret (3 cycles wasted) 49.halt

i=49 , x=1+1+1+1+2+3+3=12

⇨ Total cycles = 49+4+12=65

Below is the pipe line execution diagram of first 20 instructions each column represents one cycle and row indicates instruction

1	F	D	E	M	W																				
2		F	D	E	M	W																			
3			F	D	E	M	W																		
4				F	D	E	M	W																	
5					F	D	E	M	W																
6						F	D	E	M	W															
7							F	D	E	M	M														
8								F	D	E	E	W													
9									F	D	D	M	W												
10										F	F	E	M	W											
11											F	D	E	M	W										
12												F	D	E	M	W									
13												*	F	D	D	E	M	W							
14														F	F	D	E	M	W						
15																F	D	E	M	W					
16																	F	D	E	M	W				
17																		F	D	E	M	W			
18																			F	D	E	M	W		
19																				F	D	E	M	W	
20																					F	D	E	M	W

Task 7:

New instruction: crmmovXX rA,D(rB)

This instruction conditionally moves the value of register rA to the memory location (rB+D) by using the CC of previous instruction

Since memory control block doesn't have Cnd as input so we should modify the control block so that the following happens

M[rB+D]←Cnd?valA

I,e the valA is pushed to the memory address only if the value of Cnd is ‘1’ if the value of Cnd is ‘0’ then there is no modification to memory

Where Cnd=cond(CC,ifun)

Also this doesn’t modify the CC

All other stages work similar to cmovXX