Assignment 2

Instruction Assigned: 2 (p[OPq])

Task 1:

Instruction Encoding for p[OPq]:

Instruction	icode	ifun
paddq	C	0
psubq	C	1
pandq	C	2
pxorq	C	3

p[OPq] is a 1 Byte instruction which adds the topmost 2 values of the stack and pushes the result back onto the stack.

Task 2:

We use the paddq instruction to repeatedly add two values from top of the stack. First we push the first 2 elements of array onto stack and use paddq which pushes the sum onto the stack. Then we pop the sum into %r12 and the remaining elements which are not needed anymore. We then run a loop for the remaining 6 elements and repeat the same thing i.e. push previous sum and next element of array and then use paddq and then pop these values until we get the final sum. After that we store the final sum in %eax and return from the function.

Code:

```
.pos    0
irmovq  stack, %rsp                       # move 0x200 to stack pointer
call main                                 # call main
halt                                      # halt

.align 8
array:                                    # declaring an array of 8 elements
    .quad 0x1                             # each element is 8 byte
    .quad 0x2
    .quad 0x3
    .quad 0x4
    .quad 0x5
    .quad 0x6
    .quad 0x7
    .quad 0x8

main:
    irmovq  array,  %rdi                  # move the base address of array in
%rdi
    irmovq  $6,     %rsi                  # move iterator i in %rsi
    call sum                              # call sum
    ret                                   # return from main function

sum:
    irmovq  $8,     %r8                   # move constant 8 in %r8
    irmovq  $1,     %r9                   # move constant 1 in %r9
    mrmovq  (%rdi), %r10                  # move array[0] in %r10
    addq    %r8,    %rdi                  # increment base address of array
```

```
    mrmovq  (%rdi), %r11                    # move array[1] in %r11
    pushq   %r10                            # push array[0] onto stack
    pushq   %r11                            # push array[1] onto stack
    paddq                                   # use paddq which pushes result onto
stack
    popq    %r12                            # pop result from the stack in %r12
    popq    %r11                            # pop a[1] in %r11 (since it is not
needed anymore)
    popq    %r10                            # pop a[0] in %r10 (since it is not
needed anymore)
    addq    %r8,    %rdi                    # increment array
    xorq    %rax,   %rax                    # clear %rax
    andq    %rsi,   %rsi                    # set cc to check i
    jmp     test                            # goto test

loop:
    mrmovq  (%rdi), %r10                    # move array[8-i] in %r10
    pushq   %r10                            # push array[8-i] onto stack
    pushq   %r12                            # push previous sum onto stack
    paddq                                   # use paddq which pushes the sum onto
stack
    popq    %r12                            # pop sum from the stack in %r12
    popq    %r11                            # pop previous sum (since its not
needed anymore)
    popq    %r10                            # pop array[8-i] (since its not needed
anymore)
    addq    %r8,    %rdi                    # increment array
    subq    %r9,    %rsi                    # decrement i
    andq    %rsi,   %rsi                    # set cc to check i
    jmp     test                            # goto test

test:
    jne loop                                # if i != 0 then goto loop
    rrmovq  %r12,   %rax                    # store the final sum in %rax
    ret                                     # return from sum function

.pos 0x200
stack:
```

Task3:

Memory Dump:

```
0x0000:                         | .pos 0
0x0000: 30f40002000000000000    | irmovq stack, %rsp
0x000a: 805800000000000000      | call main
0x0013: 00                      | halt

0x0014:                         | .align 8
0x0018:                         | array:
0x0018: 01                      |   .quad 0x1
0x0020: 02                      |   .quad 0x2
0x0028: 03                      |   .quad 0x3
0x0030: 04                      |   .quad 0x4
0x0038: 05                      |   .quad 0x5
0x0040: 06                      |   .quad 0x6
0x0048: 07                      |   .quad 0x7
0x0050: 08                      |   .quad 0x8

0x0058:                         | main:
0x0058: 30f71800000000000000    |   irmovq array, %rdi
0x0062: 30f60600000000000000    |   irmovq $6,    %rsi
0x006c: 807600000000000000      |   call sum
0x0075: 90                      |   ret

0x0076:                         | sum:
0x0076: 30f80800000000000000    |   irmovq $8,    %r8
0x0080: 30f90100000000000000    |   irmovq $1,    %r9
0x008a: 50a70000000000000000    |   mrmovq (%rdi),%r10
0x0094: 6087                    |   addq %r8, %rdi
0x0096: 50b70000000000000000    |   mrmovq (%rdi),%r11
0x00a0: a0af                    |   pushq %r10
0x00a2: a0bf                    |   pushq %r11
0x00a4: c0                      |   paddq
0x00a5: b0cf                    |   popq    %r12
0x00a7: b0bf                    |   popq    %r11
0x00a9: b0af                    |   popq    %r10
0x00ab: 6087                    |   addq    %r8,    %rdi
```

```
0x00ad: 6300                    |   xorq    %rax,   %rax
0x00af: 6266                    |   andq    %rsi,   %rsi
0x00b1: 70de00000000000000      |   jmp     test

0x00ba:                         | loop:
0x00ba: 50a70000000000000000    |   mrmovq  (%rdi), %r10
0x00c4: a0af                    |   pushq   %r10
0x00c6: a0cf                    |   pushq   %r12
0x00c8: c0                      |   paddq
0x00c9: b0cf                    |   popq    %r12
0x00cb: b0bf                    |   popq    %r11
0x00cd: b0af                    |   popq    %r10
0x00cf: 6087                    |   addq    %r8,    %rdi
0x00d1: 6196                    |   subq    %r9,    %rsi
0x00d3: 6266                    |   andq    %rsi,   %rsi
0x00d5: 70de00000000000000      |   jmp     test

0x00de:                         | test:
0x00de: 74ba00000000000000      |   jne loop
0x00e7: 20c0                    |   rrmovq  %r12,   %rax
0x00e9: 90                      |   ret

0x200:                          | .pos 0x200
0x200:                          | stack:
```

Task 4:

Here, we are implementing paddq instruction in 4 cycles as follows:

1. The first cycle gets the topmost element and stores it in %r13.
2. The second cycle gets the next topmost element and stores it in %r14.
3. The 3rd cycle adds these 2 values and stores it in %r14.
4. The 4th cycle pushes the sum that is stored in %r14 to the top of the stack.

We assume that the program doesn't use %r13 and %r14 registers.
Since it requires 4 cycles, we assume that the PC is updated at the 4th cycle only.

Cycle 1:

Stage	General p[OP]q	paddq at 0x00a4
Fetch	icode:ifun ← M_1[PC] valP ← PC + 1	Icode:ifun ← M_1[PC] = C:0 valP ← PC +1 = 0x00a5
Decode	valA ← R[%rsp]	valA ← R[%rsp] = 0x01E0
Execute	valE ← valA + 0	valE←valA + 0 = 0x01E0
Memory	valM ← M_8[valE]	valM ← M_8[valE] = 2
Write Back	R[%r13] ← valM	R[%r13] ← valM = 2
PC Update	Not Updated	Not updated

Cycle 2:

Stage	General p[OP]q	paddq at 0x00a4
Fetch	icode:ifun ← M_1[PC] valP ← PC + 1	Icode:ifun ← M_1[PC] = C:0 valP ← PC +1 = 0x00a5
Decode	valA ← R[%rsp]	valA ← R[%rsp] = 0x01E0
Execute	valE ← valA + 8	valE ← valA + 8 = 0x01E8
Memory	valM ← M_8[valE]	valM ← M_8[valE] = 1
Write Back	R[%r14] ← valM	R[%r14] ← valM = 1
PC Update	Not Updated	Not updated

Cycle 3:

Stage	General p[OP]q	paddq at 0x00a4
Fetch	icode:ifun ← M_1[PC] valP ← PC + 1	Icode:ifun ← M_1[PC] = C:0 valP ← PC +1 = 0x00a5
Decode	valA ← R[%r13] valB ← R[%r14]	valA ← R[%r13] = 2 valB ← R[%r14] = 1
Execute	valE ← valB OP valA	valE ← valB + valA = 3
Memory		
Write Back	R[%r14] ← valE	R[%r14] ← valE = 3
PC Update	Not updated	Not updated

Cycle 4:

Stage	General p[OP]q	paddq at 0x00a4
Fetch	icode:ifun ← M_1[PC] valP ← PC + 1	icode:ifun ← M_1[PC] = C:0 valP ← PC + 1 = 0x00a5
Decode	valA ← R[%r14] valB ← R[%rsp]	valA ← R[%r14] = 3 valB ← R[%rsp] = 0x01E0
Execute	valE ← valB + (-8)	valE ← valB - 8 = 0x01D8
Memory	M_8[valE] ← valA	M_8[valE] ← valA = 3
Write Back	R[%rsp] ← valE	R[%rsp] ← valE = 0x01D8
PC Update	PC ← valP	PC ← valP = 0x00a5

Task 5:

Cycle ID	PC	CC Registers			Stack Pointer	Changes to memory		Changes to Registers						
		ZF	SF	OF		Address	New Value	%rdi	%rsi	%r8	%r9	%r10	%r11	%r12
1	0x0000	0	0	0	0			0	0	0	0	0	0	0
2	0x000a	0	0	0	0x0200			0	0	0	0	0	0	0
3	0x0058	0	0	0	0x01F8	0x01F8	0x0013	0	0	0	0	0	0	0
4	0x0062	0	0	0	0x01F8			0x0018	0	0	0	0	0	0
5	0x006c	0	0	0	0x01F8			0x0018	6	0	0	0	0	0
6	0x0076	0	0	0	0x01F0	0x01F0	0x0075	0x0018	6	0	0	0	0	0
7	0x0080	0	0	0	0x01F0			0x0018	6	8	0	0	0	0
8	0x008a	0	0	0	0x01F0			0x0018	6	8	1	0	0	0
9	0x0094	0	0	0	0x01F0			0x0018	6	8	1	1	0	0
10	0x0096	0	0	0	0x01F0			0x0020	6	8	1	1	0	0
11	0x00a0	0	0	0	0x01F0			0x0020	6	8	1	1	2	0
12	0x00a2	0	0	0	0x01E8	0x01E8	1	0x0020	6	8	1	1	2	0
13	0x00a4	0	0	0	0x01E0	0x01E0	2	0x0020	6	8	1	1	2	0
14	0x00a4	0	0	0	0x01E0			0x0020	6	8	1	1	2	0

15	0x00a4	0	0	0				0x0020	6	8	1	1	2	0
16	0x00a4	0	0	0				0x0020	6	8	1	1	2	0
17	0x00a5	0	0	0	0x01D8	0x01D8	3	0x0020	6	8	1	1	2	0
18	0x00a7	0	0	0	0x01E0			0x0020	6	8	1	1	2	3
19	0x00a9	0	0	0	0x01E8			0x0020	6	8	1	1	2	3
20	0x00ab	0	0	0	0x1F0			0x200	6	8	1	1	2	3

Task 6:

In the pipelined implementation of the Y86 architecture (section 4.4 - 4.5 of the book). Find the number of cycles required by your program to execute. Draw the pipelined cycle diagram for the first 20 instructions in your program.

We assume forwarding to avoid data hazards and a combination stalling & forwarding to avoid load/use and control hazards.
For conditional jump, we always assume that the branch is being taken. (PC Prediction)

No of cycles required to execute for the whole Program: 147 cycles

Instruction	Cycle No: 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
1 irmovq stack,%rsp	F	D	E	M	W																				
2 call main		F	D	E	M	W																			
3 irmovq array,%rdi			F	D	E	M	W																		
4 irmovq $6,%rsi				F	D	E	M	W																	
5 call sum					F	D	E	M	W																
6 irmovq $8,%r8						F	D	E	M	W															
7 irmovq $1,%r9							F	D	E	M	W														
8 mrmovq (%rdi),%r10								F	D	E	M	W													
9 addq %r8,%rdi									F	D	E	M	W												
10 mrmovq (%rdi),%r11										F	D	E	M	W											
11 pushq %r10											F	D	E	M	W										
12 pushq %r11												F	D	E	M	W									
13 paddq													F	D	E	M	W								
14 paddq														F	D	E	M	W							
15 paddq															F	D	E	M	W						
16 paddq																F	D	E	M	W					
																			E	M	W				
17 popq %r12																	F	D	D	E	M	W			
18 popq %r11																		F	F	D	E	M	W		
19 popq %r10																			F	F	D	E	M	W	
20 addq %r8,%rdi																				F	F	D	E	M	W

We observe that most of the data/load/use hazards are prevented by forwarding.
At cycle 19, We use a bubble to avoid load/use hazard since paddq needs to pass the execute stage to forward the value being return at the top of the stack to popq %r12. After it reaches memory stage, M_valM is forwarded to the decode stage of popq %r12 so that it can read the correct value of the top of the stack which paddq is going to push.

Rest everywhere, forwarding controls the data hazard.

Task 7:

New instruction: irmulq $imm, %rA

This instruction multiplies the immediate value with the value in %rA and stores the result in %rA. If the result is more than 8 bytes(64 bits) then the upper value is stored in %rdx. It will require a specialised multiplication unit as the existing ALU can't do the multiplication in a single clock cycle. %rA can be any of the registers except %rdx.