Name: PRAKRAM RATHORE
Roll No: 20110141

ES215: Assignment - 2

Q1

Ans → In C++, a function can return almost 1 variable/value. So, we can't write a single function returning both highest value and index. So I'll be writing a function, which will return the maximum value index. And, for value we can find A[i] directly from main function.

Function to find highest element in an array a[100]

```
int highest (int a[], int n){
        int max= -2147483648;
        int index = 0;
        int i = 0;
        while (i<n){
              if (a[i] >temp){
                 max = a[i] ;
                    index = i ;
              }
              i++;
        }
        return index;
}
```

We are getting base address of Array and size of array as argument

∴ $\boxed{\$a_0 = \text{base address of array } a}$, $\boxed{\$a_1 = n.}$

highest:

```
        addi  $sp , $sp , -16        # Shift down $sp by 16
        sw    $s0 , 0($sp)           # Saving Callee saved
        sw    $s1 , 4($sp)             registers
        sw    $s2 , 8($sp)
        sw    $ra , 12($sp)
        addi  $s0 , $0   INT_MIN     # $s0 = max [INT_MIN = -2147483648]
        addi  $s1 , $0   0           # $s1 = index
        addi  $s2 , $0   0           # $s2 = i
                                       (While loop)
while:
        beq   $s2 , $a1 , exit       # If i==n, go to exit
        lw    $t1 , 0($a0)           # $t1 = base address of array
        add   $t1 , $t1 , $s2        # $t1 = address of ai
        lw    $t0 , 0($t1)           # $t0 = Value of ai
        slt   $t2 , $s0 , $t0        # $t2=1 if max < ai
        bne   $t2 , $0 , do          # if t2==1, go to do

do:     add   $s0 , $0 , $t0         # max = a[i]
        add   $s1 , $0 , $s2         # index = i
        j     Continue               # Jump to Continue

Continue:
        addi  $s2 , $s2 , 1          # i++
        j     while.                 # jump to while
```

exit:

(Exit)

```
lw    $v0 , 0($s1)       # loading return value
lw    $s0 , 0($sp)       # Loading all callee saved
lw    $s1 , 4($sp)         registor's value before returning
lw    $s2 , 8($sp)
lw    $ra , 12($sp)
addi  $sp , $sp , 12     # Incrementing $sp [stack frame remove]
jr    $ra.               # Return.
```

Note! For Max^m & Min^m values, we only need there resp. indices, the we can easily call a[i] in main function. & that can be done by add offset value i to base address of a.

```
addi  $a0  $a0  i
ld    $t0  0($a0)
```

So, we only need to return index from func^s.

For lowest Element; [$a0 = base address of array, $a1 = size of array]

```
int lowest (int a[], int n){
        int min = 2147483647;
        int index = 0;
        int i = 0;
        while (i<n){
              if (a[i] < min){
                    min = a[i];
                    index = i;
              }
              i++;
        }
        return min;
}
```

Assembly Code:

```
lowest:
     addi  $sp , $sp , -12        # shift down $sp by 12
     sw    $s0 , 0($sp)           # saving callee saved
     sw    $s1 , 4($sp)             registers
     sw    $s2 , 8($sp)
     addi  $s0 , $0 , INT_MAX     #$s0 = min
     addi  $s1 , $0 , 0           # $s1 = index
     addi  $s2 , $0 , 0           # $s2 = i
while :                           (while loop)
     beq   $s2 , $a1 , exit       # If i==n go to exit
     lw    $t1 , 0($a0)           # $t1= base address of array a.
     add   $t1 , $t1 , $s2        # $t1 = address of ai
     lw    $t0 , 0($t1)           # $t0 = value of ai
```

```
    slt   $t2 , $t0 , $s0      # $t2 = 1 if ai < min
    bne   $t2 , $0 , do        # if t2 == 1, go to do

do:
    add   $s0 , $0 , $t0       # min = a[i]
    add   $s1 , $0 , $s2       # index = i
    j     Continue             # Jump to continue

Continue:
    addi  $s2 , $s2 , 1        # i++
    j     while                # jump to while

exit:                          (exit)
    lw    $v0 , 0($s1)         # Loading return value
    lw    $s0 , 0($sp)         # Loading all calee saved
    lw    $s1 , 4($sp)           register's value before return
    lw    $s2 , 8($sp)
    addi  $sp , $sp 12         # Incrementing $sp (stack frame remove)
    jr    $ra                  # Return.
```

for Average

```
int average (int a[], int n){
    int sm = 0;
    int i = 0;
    while (i<n) {
        sm = sm + a[i];
        i++;
    }
    return sm/n;
}
```

Assembly Code

```
average:
    addi $sp, $sp, -8      # shift down $sp
    sw   $s0, 0($sp)       # Saving callee saved
    sw   $s1, 4($sp)       #   registers
    addi $s0, $0, 0        # s0 = sm
    addi $s1, $0, 0        # s0 = i
while:
    beq  $s1, $a1, exit    # If i==n, go to exit
    lw   $t1, 0($a0)       # t1 = base address of array a
    add  $t1, $t1          # t1 = address of ai
    lw   $t0, 0($t1)       # t0 = value of ai
    add  $s0, $s0, $t0     # sm = sm + a[i]
    addi $s1, $s1, 1       # i++
    j    while             # Jump to while
exit:
    div  $s0, $s0, $a1     # divide sm/n
    lw   $v0, 0($sp)       # Loading return value
    lw   $s0, 0($sp)       # Loading callee saved registers
    lw   $s1, 4($sp)
    jr   $ra               # Return.
```

Q2

→ A program A will take 6 seconds on Core 1 with CPI = 6

and Same program A will take 5 seconds on Core 2 with CPI = 5.

→ Both cores runs at a clk rate of 1 GHz

→ We need to find combined throughput of Processors

We know that throughput of any system is the total work done per unit time. Here, in CPU Processors, it is the total no. of Instruction done in 1 sec.

① For Core 1,

~~Total no. of Instruction = CPI × IC = C~~

$$\text{Total time} = \frac{CPI \times IC}{CR}$$

$$\text{Instruction count} = I_1 = \frac{6 \times 10^9}{6} = \boxed{10^9}$$

② For Core 2,

$$\text{Total time} = \frac{5 \times I.C \times 5}{10^9}$$

$$\therefore \text{Instruction Count} = I_2 = \boxed{10^9}$$

So, Combined, Total of $I_1 + I_2$ instructions have been done in Total of 6 Seconds, [Both are happening in parallel]

So $t = max(t_1, t_2)$ $\therefore I_1 + I_2 = 2 \times 10^9$ instructions

$\therefore$ $t = 6$ seconds

$\therefore$ Combined throughput $= \frac{I_1 + I_2}{t}$

$= \frac{2 \times 10^9}{6}$

$\Rightarrow 0.33333 \times 10^9$

$\boxed{\text{Combined Throughput} \Rightarrow 333.33 \times 10^6 \ \frac{\text{Instructions}}{\text{Second}}}$

Q3

For Processor-X which runs at 2 GHz.

for Programme A:

No. of Instructions = 10 billion

avg. CPI = 3

$\therefore t_x = \dfrac{\text{Instruction} \times \text{CPI}}{\text{Rate}}$

$= \dfrac{10 \times 10^9 \times 3}{2 \times 10^9} = 15$

$\therefore \boxed{t_x = 15}$

For Processor-Y which runs at 4 GHz

for Programme A:

No of Instructions = 7 billion

avg. CPI = 5

$\therefore t_y = \dfrac{\text{Instructions} \times \text{CPI}}{\text{Rate}} = \dfrac{7 \times 10^9 \times 5}{4 \times 10^9} = 8.75$

$\therefore \boxed{t_y = 8.75}$

Speedup for A on processor Y over processor X

Speedup $(Y/X) = t_x/t_y = \dfrac{15}{8.75} \Rightarrow 1.714$

Q4 Processor has Rate of 1 GHz
runs Programm A, ~~while~~ with

9 billion instructions

avg. CPI = 1.5

$$t = \frac{9 \times 10^9 \times 1.5}{10^9} \Rightarrow 13.5$$

Now, new Processor with Rate = 2 GHz

$$t' = t/4 = \frac{13.5}{4} = \frac{9 \times 10^9 \times x}{2 \times 10^9}$$

$$x = 0.75$$

$$\boxed{(\text{avg. CPI})' = 0.75}$$

Q5

We have not given the proportion of actual dynamic & static power.

Let Ratio of Dynamic & Static power be $t:1$

$\therefore$ Dynamic Power $= \left(\frac{t}{t+1}\right) 80$ watts

Static Power $= \left(\frac{1}{t+1}\right) 80$ watts

Given, frequency $= 2$ GHz and operating voltage $= 5V$

a) If frequency becomes 5 GHz $\Rightarrow$

We know that dynamic power $= \frac{1}{2} C V^2 f \Rightarrow K f$

where $C =$ capacitive Load
$V =$ Voltage
$f =$ freq.

$$\frac{\text{Initial Dynamic Power}}{\text{final Dynamic Power}} = \frac{\cancel{K}\, 2}{\cancel{K}\, 5} = 2/5$$

$\therefore$ final dynamic power $= \frac{5}{2} \times \frac{t}{t+1} \times 80$

$$= \boxed{\frac{200t}{t+1} \text{ watts}}$$

b) Voltage changes to 2V.

We know that Static power = IV,

I = Leakage current

V = Voltage

We also know that $I_{leakage}$ is almost constant

$\therefore$ Static power $\propto V$

$$\therefore \frac{(\text{Static Power})_i}{(\text{Static Power})_f} = 5/2$$

$\therefore$ final static Power $= \frac{32}{t+1}$ watts

$$\frac{(\text{Dynamic Power})_i}{(\text{Dynamic Power})_f} \Rightarrow \frac{(2)^2}{(5)^2} = 4/25$$

final Dynamic Power $= \left(\frac{t}{t+1} \times 80\right) \times 4/25$.

$\therefore$ Total Power = Dynamic Power + Static Power

$$= \frac{32}{t+1} + \frac{12.8t}{t+1}$$

$$\boxed{\text{Total Power} = \frac{32+12.8t}{t+1}}$$

$$\boxed{\% \text{ of Static Power} \Rightarrow \frac{12.8t \times 100}{32+12.8t}}$$

$$\boxed{\text{fraction} = \frac{12.8t}{32+12.8t}}$$

According to, book => Computer Organization & Design By DAVID A. PATTERSON & JOHN L. HENNESY, Page No:-42)

Even when a server is off, some leakage current is still present & that constitutes on an average 40%, $\therefore$

On an average Static Power is 40% Total power, $\therefore$ We may take $[t \Rightarrow 3/2]$

Grace Question

b) (1) fib_x86.i [Preprocessed]

Size: 3.5 MB

Observation

- Structure ⇒ {
  templates
  classes
  Object
  }

include /x86-64-linux-gnu/c++/11/bits/stdc++.h"

using namespace std.

Rest is the Same Code. {

- Various kind of template, classes were written after that x86-64-linux-gnu/c++/11/bits/stdc++.h was included

(2) fib_mips.i [Preprocessed]

Size: 2.9 MB

Same Structure except the templates & classes were different also in mips /mips-linux-gnu/c++/10/mips-linux-gnu/bits/stdc++.h was included.

(3) fib_x86.obj [Assembled]

Size: 5.6 KB

Observation →

It was a kind of paragraph made by some symbols like ⟨?⟩.

That single paragraph was also not readable. that was the way we've studied in class.

(4) fib_mips.obj [Assembled]

Size: 3.7 KB

Observation →

Same as fib_x86.obj, it was not readable.

(5) fib_x86 out [Binary Code]

Size: 18.1 Kb

(6) fib_~~x86~~ mips.out [Binary code]

Size: 8.7 KB

→ Both were same as above in appearance, not readable.

(7) fib_x86.s [Compiled code]

Size: 9KB

Observation :

It was huge code starting with something

-ZN6 -pst 9_ ______

______ ______

• zero 1
• local
..

______ ______

→ Several times

Then there was actual ~~ASS~~ compiled code like we've studied in class in MIPS, the terms were different as it is <u>x86</u>.

→ In place of $ → %

Several mnemonic :- movq, subq, xorl, leaq etc.

(8) <u>fib mips.s</u> [compiled code]

Size :- 6.4 KB.

<u>Observation</u>

It was almost same as we've done in class. Although, Not whole code was understandable, but main part was clear, various chunk of sw, lw was there showing the process of saving & loading caller & callee saved registers values. Jalr was there & by focus I was able to understand the main crux of code.