Assignment-2

https://github.com/Madhav-Kanda/MadhavKanda_ES215_Assignment2

1. **Assumptions:**
    - The inputs are such that the average is always an integer as the div operation used will always give value in terms of integer.
    - The size of the array has been directly stored in the variable value ($t8) as 100.
    - The given array is pre-filled with the values.

MIPS Code

main:

```
addi $t0, $0, 200          # putting the base address of the array in $t0
lw $t1, 0($t0)             # Highest value of the array = $t1= array[0]
lw $t2, 0($t0)             #Lowest value of the array = $t2 = array[0]
addi $t3,$0,0              #Highest value Index = $t3 =0
addi $t4,$0,0              # Lowest value Index = $t4 =0
addi $t6,$0,0              #$t6=sum=0
addi $t7,$0,0              #$t7=index=0
addi $t8,$0,100            #t8= value=100
```

while:

```
beq $t7,$t8,average
lw $s2,0($t0)              #load the value in $s2
slt $s1,$t1,$s2            # if $t1 less than $s2 then $s1 will become 1
bne $s1,$0, Highest        #if the value of $s1 not equals 0 then move to Highest
slt $s1,$s2,$t1            # if $s2 less than $t1 then $s1 will become 1
bne $s1,$0, Lowest         #if the value of $s1 not equals 0 then move to Lowest
addi $t7,$t7,1             #index+=1
addi $t0,$t0,4             #incrementing the index value of the pointer
add $t6,$t6,$s2            #increment sum
j while
```

Highest:

```
add $t1,$0,$s2             # Highest = array[0]
```

```
      add $t3,$0,$t7           #Highest Index= current index
      addi $t7,$t7,1           #index+=1
      add $t6,$t6,$s2          #increment sum
      addi $t0,$t0,4           #incrementing  index value of the pointer
      j while

Lowest:
      add $t2,$0,$s2           # Lowest = array[0]
      add $t4,$0,$t7           #LowestInd= current index
      addi $t7,$t7,1           #index+=1
      add $t6,$t6,$s2          #increment sum
      addi $t0,$t0,4           #incrementing the index value
      j while

average:
      div $t6,$t6,100          #Dividing by 100
      jr $ra
```

2. **Throughput = (Total number of instructions/Time taken to complete the instructions)**
   Total time = (CPI)*(Instruction Count)/(Clock Rate)
   Therefore,
   Instruction Count of Program A = (Total time) * (Clock Rate)/(CPI) = 6*(10^9)/6 = **10^9**
   Instruction Count of Program B = (Total time) * (Clock Rate)/(CPI) = 5*(10^9)/5 = **10^9**

   Total Instruction count = Instruction count of Program A + Instruction count of Program B = **2*(10^9)**

   Total time taken to complete the instruction count = maximum of (time taken by A, time taken by B) {This is because the processes are being done in different cores and therefore parallel operation} = **6 seconds**

   Thus, Throughput = (Total number of instructions/Time taken to complete the instructions) = 2*(10^9)/6 = 10^9/3 = 3.33*10^8 Instructions per second

3. **Total time taken = (CPI* Instruction count)/ (Clock Rate)**

Given Instruction count of processor X for program A = 10*(10^9)
Given Clock Rate of processor X for program A = 2*(10^9) Hz
Given avg. CPI of processor X for program A = 3
Total time taken by processor X = 10*(10^9)*3/(2*(10^9) = **15 seconds**

Given CPI of processor Y for program A = 7*(10^9)
Given Clock Rate of processor Y for program A = 4*(10^9) Hz
Given avg. CPI of processor Y for program A = 5
Total time taken by processor Y = 7*(10^9)*5/(4*(10^9) = **8.75 seconds**

Speed up of Program A on processor Y over the processor X = total time taken by processor X/ total time taken by processor Y = (15/8.75) = 1.71428

Therefore, Processor Y is 1.71428 times faster than Processor X for program A

4. **Total time taken = (CPI* Instruction count)/ (Clock Rate)**
Given Instruction count for program A = 9*(10^9)
Given Clock Rate for program A = 1*(10^9) Hz
Given avg. CPI for program A = 1.5
Total time taken by program A on the processor = 9*(10^9)*1.5/(1*(10^9) = **13.5 seconds**

Total time taken by program A on the newly designed processor = total time taken by program A on the previous processor/4 = 13.5/4 = **3.375 seconds**

Assuming that the ISA remains the same and thus the Instruction count for the program remains the same on the new processor design

Thus, 3.375 seconds = (CPI of program A on new design* Instruction count)/(Clock rate of the new processor)

Thus the CPI of the program for the new processor = (3.375* clock rate of the new processor)/ Instruction count = 3.375*(2*10^9)/(9*10^9) = **0.75**

Therefore, the CPI of the program on the new processor design is 0.75 counts per instructions

5. **Here to solve the question I have assumed that the dynamic power constitutes 60% and thus the static power constitutes 40% of the total power. This has been taken according to what has been mentioned in the book**

Dynamic power is given by the formula = **0.5*C(V^2)*f**
Here C represents the Capacitance, V represents the voltage and f represents the frequency

Static power is given by the formula = VI
Given total power = 80W

Therefore, according to the assumption, Static power will be 0.4*80 = 32W
Dynamic Power will be 0.6*80 =48W

a) Now if the frequency gets increased to 5GHz then the Dynamic Power will become 5/2 times the current dynamic power as the dynamic power is directly proportional to the frequency. Since the earlier frequency was 2 GHz thus the new value will be 5/2 times the current value as it has increased to 5 GHz thus the value of the dynamic power is: 48*2.5 W = **120 W**
b) Now the operating value of the voltage drops to 2V.
   Static power = VI = 32*(2/5) = **12.8W**
   Dynamic power = 0.5*C(V^2)*f = (48*(2/5)^2) = **7.84W**
   Overall power consumption = static power + dynamic power = 12.8W +7.84W = 20.64W

   Fraction of static power in the total power = static power/total power = (12.8/20.64) = 0.6201

Grace Question

1. **The pre-processor, compiler, and assembler generated files of each of mips and x_86 have been stored in the folders named mips and x_86 respectively.**

Observations:

- **.**i file for mips (51 kb) is greater than that for the x86 files(48 kb) (Pre-processor)
- .o file for mips(2.32 kb) is less than that for the x86 files(2.72 kb) (assembler)
- .s files for mips(3.73 kb) is greater than that for x86 files(2.69 kb) (compiler)
- .out file for mips(255 kb) is significantly greater than x86 files(49 kb) (Binary)

Github link: https://github.com/Madhav-Kanda/MadhavKanda_ES215_Assignment2