Homework #1

Assigned: 19/03/2020
Due: 27/03/2020

1. (**20 pts**) Consider the following table that describes an implementation of an instruction set architecture and the occurrence frequency of the instructions in a given benchmark.

Instruction Class	clock cycles (CPI)	Frequency
A	5	25%
B	6	10%
C	2	20%
D	8	10%
E	4	35%

a. Compute the overall CPI for the given benchmark. (**5 pts**)

CPI = CPI_A * relative_freq_A + CPI_B * relative_freq_B + CPI_C * relative_freq_C + CPI_D * relative_freq_D + CPI_E * relative_freq_E
=5 * 25/100 + 6 * 10/100 + 2 * 20/100 + 8 * 10/100 + 4 * 35/100
= 445 / 100 = 4.45

b. The clock frequency is 3.2 GHz and the number of instructions in the benchmark is 15 billion. Compute the execution time of the benchmark. (**5 pts**)

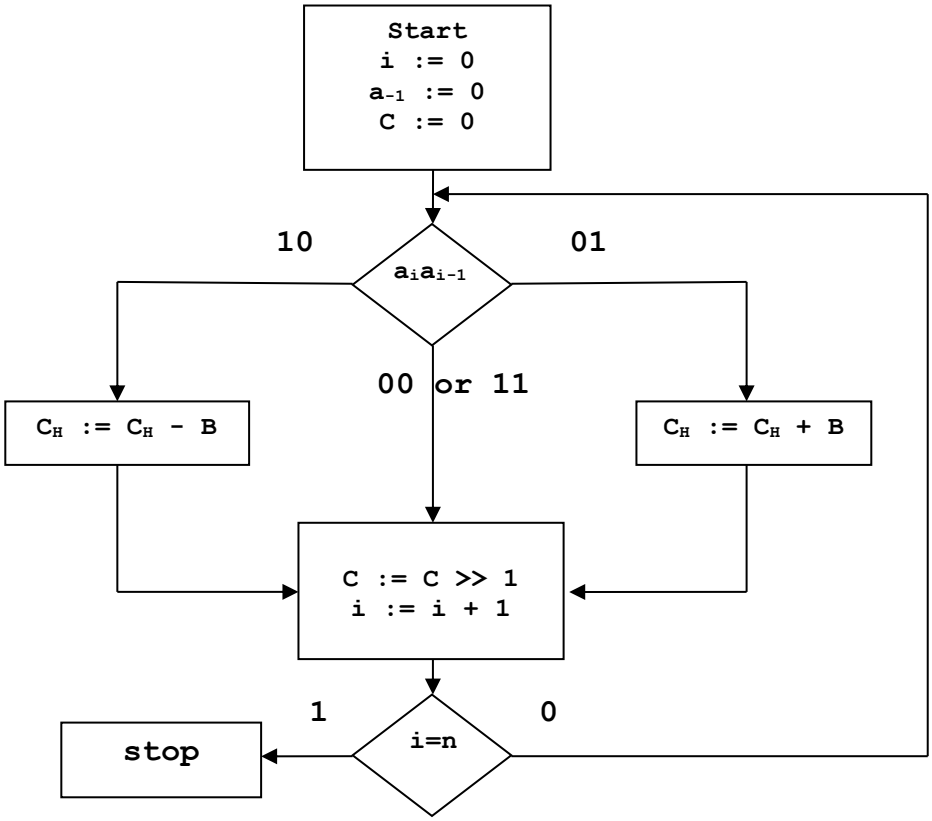
T = number of instructions * clock cycles per instruction * clock cycle time
= 15 * 10^9 * 4.45 * 1 / (3.2 x 10^9)
= 20.859375 seconds

c. Assume that you are given an opportunity of reducing the CPI of any one class of instruction to 1. You can select at most one instruction to improve. Which instruction class would you choose and how much can the performance be improved? (**10 pts**)

I would choose class E because it gives the most impact (by calculating CPI * frequency) when its CPI is set to 1, compared to other classes.
If CPI of E was 1, overall CPI would be:
CPI = 5 * 25/100 + 6 * 10/100 + 2 * 20/100 + 8 * 10/100 + 1 * 35/100 = 3.4
Considering performance as 1/execution time and execution time = number of instructions * CPI * clock cycle time, instruction count and clock cycle time does not change, performance is reversely proportional to CPI.

4.45/3.4 ≅ 1.30882, 30.88% improvement approximately.

2. (**30 pts**) Consider Booth's algorithm below for multiplying integers including signed ones (two' complement).



$C = A \times B$, C is $2n$-bit, A and B are n-bit registers. C_H is upper n-bit of C register.

Using Booth's algorithm with $n = 4$, do the following multiplication operations:

$6 \times 5 = ?$
$-5 \times -4 = ?$
$-4 \times 7 = ?$ (**10 pts each**)

Shows the steps of the algorithm.

6 × 5

i	B	A	$a_i a_{i-1}$	operation	C
0	0101	0110	00	C := C >> 1	00000000
1	0101	0110	10	C_H := C_H - B	10110000
				C := C >> 1	11011000
2	0101	0110	11	C := C >> 1	11101100
3	0101	0110	01	C_H := C_H + B	00111100
				C := C >> 1	00011110

-5 × -4

i	B	A	$a_i a_{i-1}$	operation	C
0	1100	1011	10	C_H := C_H - B	01000000
				C := C >> 1	00100000
1	1100	1011	11	C := C >> 1	00010000
2	1100	1011	01	C_H := C_H + B	11010000
				C := C >> 1	11101000
3	1100	1011	10	C_H := C_H - B	00101000
				C := C >> 1	00010100

-4 × 7

i	B	A	$a_i a_{i-1}$	operation	C
0	0111	1100	00	C := C >> 1	00000000
1	0111	1100	00	C := C >> 1	00000000
2	0111	1100	10	C_H := C_H - B	10010000
				C := C >> 1	11001000
3	0111	1100	11	C := C >> 1	11100100

3. (**30 pts**) Consider the following C language statements.

- `f = -g + h + B[i]+ C[j]` (**10 pts**)
- `f = A[B[g]+ C[h] + j]` (**20 pts**)

Assume that the local variables `f, g, h, i` and `j` of integer types (32-bit) are assigned to registers `$s0, $s1, $s2, $s3` and `$s4` respectively. Assume also the base address of the arrays `A, B` and `C` of integer types are in registers `$s5, $s6` and `$s7`, respectively (i.e. `$s0` → `f, $s1` → `g, $s2` → `h, $s3` → `i, $s4` → `j, $s5` → `&A[0] , $s6` → `&B[0], $s7` → `&C[0]`).

For the C statements above, provide MIPS Assembly instructions.

f = -g + h + B[i]+ C[j]:

```
sub $s0, $s2, $s1  #f = h – g
sll $t0, $s3, 2  #to calculate the offset, multiply i by 4
add $t0, $t0, $s6  #add 4i to the base register of array B
lw $t0, 0($t0)  #load word, B[i]
sll $t1, $s4, 2  #multiply j by 4
add $t1, $t1, $s7  #add 4j to the base register of array C
lw $t1, 0($t1)  #load word, C[j]
add $s0, $s0, $t0  #f = h – g + B[i]
add $s0, $s0, $t1   #f = h – g + B[i] + C[j]
```

f = A[B[g]+ C[h] + j]:

```
sll $t0, $s1, 2 #to calculate the offset, multiply g by 4
add $t0, $t0, $s6
lw $t0, 0($t0) #load word, B[g]
sll $t1, $s2, 2  #to calculate the offset, multiply h by 4
add $t1, $t1, $s7
lw $t1, 0($t1) #load word, C[h]
add $t0, $t0, $t1  #B[g] + C[h]
add $t0, $t0, $s4 #B[g] + C[h] + j
sll $t0, $t0, 2 #to calculate the offset, multiply B[g]+ C[h] + j by 4
add $t0, $t0, $s5
lw $s0, 0($t0) #load word, f = A[B[g]+ C[h] + j]
```

4. (**20 pts**) Consider the following C++ code sequence

```
t = A[0];
for (i=0; i < 5; i++)
     A[i] = A[i+1];
A[5] = t;
```

Write an Assembly language program for MIPS processor, assuming that base address of the array **A** is in **$s0**.

```
        lw $t0, 0($s0)  #t = A[0]
        add $t1, $zero, $zero  #i = 0
Loop:   slti $t2, $t1, 5  #$t2 = 1 if $t1 < 5, else $t2 = 0
        beq $t2, $zero, Exit  #if $t2 = 0, exit the loop
        sll $t2, $t1, 2  #to calculate the offset, 4 * I
        add $t2, $t2, $s0  #$t2 has A[i]
        addi $t3, $t1, 1  #i++, to be used
        sll $t3, $t3, 2  #to calculate the offset, 4 * (i+1)
        add $t3, $t3, $s0  #$t3 has A[i+1]
        lw $t3, 0($t3)  #load word, A[i+1]
        sw $t3, 0($t2)  #A[i] = A[i+1]
        addi $t1, $t1, 1  #i++, for the for loop
        j Loop  #continue the loop
Exit:   sw $t0, 20($s0)  #A[5] = t
```