Homework #1 Cavit Çakır 23657

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 = ((5*25/100) + (6*10/100) + (2*20/100) + (8*10/100) + (4*35/100)) = 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 = (4,45 * 15 * 10^9) / (3,2 * 10^9) = 20,8 sec

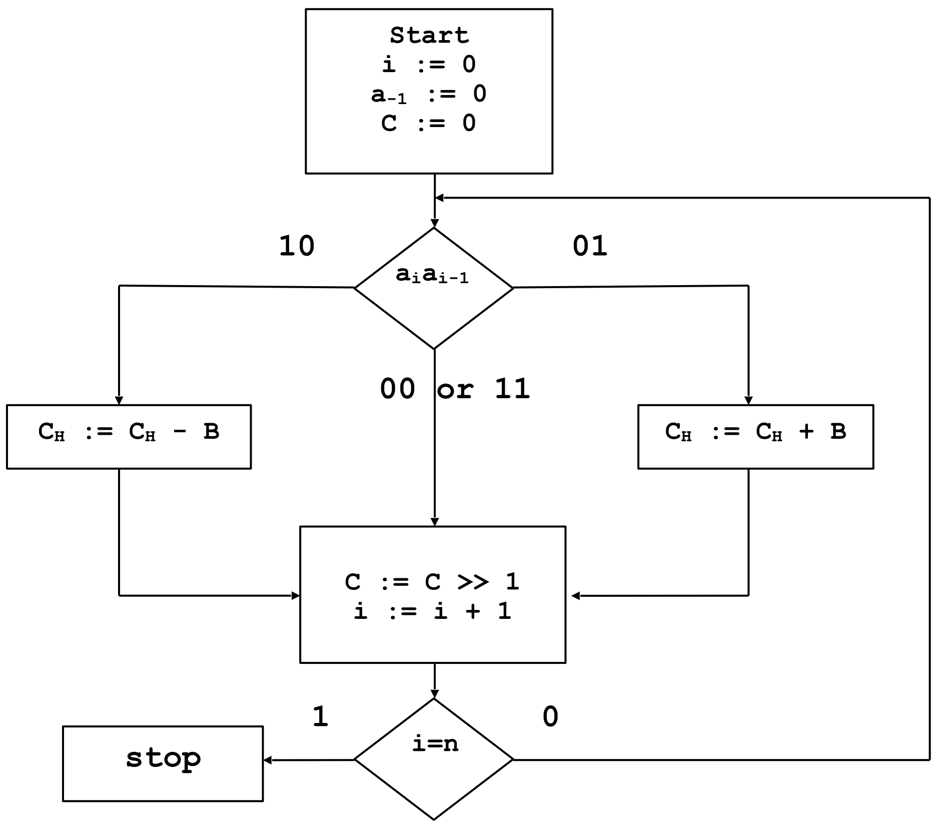
c. Assume that you are given an opportunity of reducing the CPI of any one class of instruction to 1. You can select at most one instruction to improve. Which instruction class would you choose and how much can the performance be improved? (**10 pts**)

A -> 5 * 25/100 = 1,25 —> when CPI is 1 —> 0,25 —> deducted 1
B -> 6 * 10/100 = 0,6 —> when CPI is 1 —> 0,10 —> deducted 0,5
C -> 2 * 20/100 = 0,4 —> when CPI is 1 —> 0,20 —> deducted 0,2
D -> 8 * 10/100 = 0,8 —> when CPI is 1 —> 0,10 —> deducted 0,7
E -> 4 * 35/100 = 1,4 —> when CPI is 1 —> 0,35 —> deducted 1,05

So, I should choose class E because it has most difference when CPI is set to 1.
Overall CPI will be 4,45 - 1,05 = 3,40
T will be (3,40 * 15 * 10^9) / (3,2 * 10^9) = 15,9 sec

So, performance will be improved by 20,8 / 15,9 = 1,3 which means 30%.

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_ia_{i-1}	operation	C
0	0101	0110	00	no-op	0000 0110 0
				right-shift	0000 0011 0
1	0101	0110	10	Subtract Multiplicand	1011 0011 0
				right-shift	1101 1001 1
2	0101	0110	11	no-op	1101 1001 1
				right-shift	1110 1100 1
3	0101	0110	01	add Multiplicand	0011 1100 1
				right-shift	0001 1110 0

−5 × −4

i	B	A	a_ia_{i-1}	operation	C
0	1100	1011	10	Subtract Multiplicand	0100 1011 0
				right-shift	0010 0101 1
1	1100	1011	11	no-op	0010 0101 1
				right-shift	0001 0010 1
2	1100	1011	01	add Multiplicand	1101 0010 1
				right-shift	1110 1001 0
3	1100	1011	10	Subtract Multiplicand	0010 1001 0
				right-shift	0001 0100 1

-4 × 7

i	B	A	$a_i a_{i-1}$	operation	C
0	0111	1100	00	no-op	0000 1100 0
				right-shift	0000 0110 0
1	0111	1100	00	no-op	0000 0110 0
				right-shift	0000 0011 0
2	0111	1100	10	Subtract Multiplicand	1001 0011 0
				right-shift	1100 1001 1
3	0111	1100	11	no-op	1100 1001 1
				right-shift	1110 0100 1

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.

```
Q1:
sub $s0, $s2, $s1        -> -g + h
sll $t1, $s3, 2          -> t1 -> address b[i] - s6
add $t2, $s6, $t1        -> t2 -> address b[i]
lw $t3, 0($t2)           -> t3 -> value B[i]
sll $t4, $s4, 2          -> t4 -> address c[j] - s7
add $t5, $s7, $t4        -> t5 -> address c[i]
lw $t6, 0($t5)           -> t6 -> value c[j]
add $s0, $t6, $s0        -> -g + h + c[j]
add $s0, $t3, $s0        -> -g + h + c[j] + b[i]


Q2:
sll $t1, $s1, 2          -> t1 -> address b[g] - s6
add $t2, $s6, $t1        -> t2 -> address b[g]
lw $t3, 0($t2)           -> t3 -> value B[g]
sll $t4, $s2, 2          -> t4 -> address c[h] - s7
add $t5, $s7, $t4        -> t5 -> address c[h]
lw $t6, 0($t5)           -> t6 -> value c[h]
add $t7, $t3, $t6        -> t7 -> b[g] + c[h]
add $t7, $t7, $s4        -> t7 -> b[g] + c[h] + j
sll $t7, $t7, 2          -> t7 *= 4
add $t7, $t7, $s5        -> address of a[b[g] + c[h] + j]
lw $s0, 0($t7)           -> 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]
move $t1, $zero               -> i = 0

loop:
          beq $t1, 5, exit    -> if i == 5 go exit
          sll $t2, $t1, 2     -> t2 -> address A[i] - s0
          add $t2, $s0, $t2   -> t2 -> address A[i]
          lw $t3, 4($t2)      -> t3 -> value A[i+1]
          sw $t3, 0($t2)      -> A[i] = A[i+1]
          addi $t1, $t1, 1    -> i++
          j loop              -> loop
exit:
          sw $t0, 4($t2)      -> A[5] = t
```