Efe Şencan 25083

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 = 5*(0.25) + 6*(0.1) + 2*(0.2) + 8*(0.1) + 4*(0.35) = 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**)

$T = (15 * 10^9) * (4.45) / (3.2* 10^9) = 20.859$

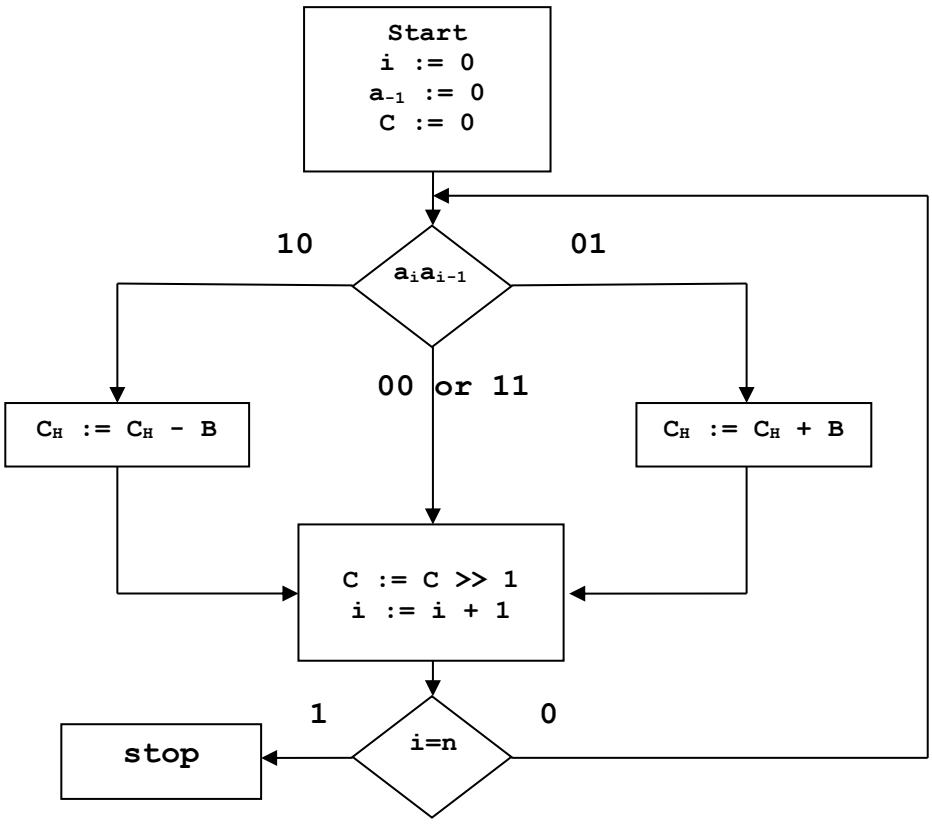
c. Assume that you are given an opportunity of reducing the CPI of any one class of instruction to 1. You can select at most one instruction to improve. Which instruction class would you choose and how much can the performance be improved? (**10 pts**)

I would choose the instruction E, because it is the most frequent instruction.

Old CPI = 4,45
Updated CPI = 5*(0.25) + 6*(0.1) + 2*(0.2) + 8*(0.1) + 1*(0.35) = 3,4
Since the clock frequency and number of instructions remain same:
Performance Improvement Rate is 4,45 / 3,4 = 1,3 ~ 30% performance improvement

I

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	No Operation	0000 0110 0
				C := C >> 1	0000 0011 0
1	0101	0110	10	$C_H := C_H - B$	1011 0011 0
				C := C >> 1	1101 1001 1
2	0101	0110	11	No Operation	1101 1001 1
				C := C >> 1	1110 1100 1
3	0101	0110	01	$C_H := C_H + B$	0011 1100 1
				C := C >> 1	0001 1110 0

−5 × −4

i	B	A	$a_i a_{i-1}$	operation	C
0	1100	1011	10	$C_H := C_H - B$	0100 1011 0
				C := C >> 1	0010 0101 1
1	1100	1011	11	No Operation	0010 0101 1
				C := C >> 1	0001 0010 1
2	1100	1011	01	$C_H := C_H + B$	1101 0010 1
				C := C >> 1	1110 1001 0
3	1100	1011	10	$C_H := C_H - B$	0010 1001 0
				C := C >> 1	0001 0100 1

−4 × 7

i	B	A	$a_i a_{i-1}$	operation	C
0	0111	1100	00	No Operation	0000 1100 0
				C := C >> 1	0000 0110 0
1	0111	1100	00	No Operation	0000 0110 0
				C := C >> 1	0000 0011 0
2	0111	1100	10	$C_H := C_H - B$	1001 0011 0
				C := C >> 1	1100 1001 1
3	0111	1100	11	No Operation	1100 1001 1
				C := C >> 1	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.

First one:

```
sub $t0, $s2, $s1  # t0 = h - g
sll  $t1, $s3, 2 # t1 = i*4
sll  $t2, $s4, 2 # t2 = j*4
add  $t3, $s6, $t1 # t3 = &B[i]
add $t4, $s7, $t2 # t4 = &C[j]
lw $t3, 0($t3) # t3 = B[i]
lw $t4, 0($t4) # t4 = C[j]
add $t5, $t0, $t3 # t5 =  -g + h + B[i]
add $s0, $t5, $t4 # f  =  -g + h + B[i] + C[j]
```

Second One:

```
sll $t0, $s1, 2 # t0 = g*4
sll $t1, $s2, 2 # t1 = h * 4
add $t3, $s6, $t0 # t3 = &B[g]
add $t4, $s7, $t1 # t4 = &C[h]
lw $t3, 0($t3) # t3 = B[g]
lw $t4, 0($t4) # t4 = C[h]
add $t5, $t3, $t4 # t5 =  B[g] + C[h]
add $t5, $t5, $s4 # t5 = B[g] + C[h] + j
sll $t6, $t5, 2 # t6 = (B[g] + C[h] + j) * 4
add $t6, $s5, $t6 # t6 = &A[B[g] + C[h] + j]
lw  $s0, 0($t6) # 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 $t, 0($s0) # t = A[0]
li   $t1, 0 # initialize loop counter as 0
loop:
beq $t1, 5, exit # exit the loop if loop counter reaches to 5
add $t2, $t1, 1 # t2 = i + 1
sll    $t2, $t2, 2 # t2 = (i+1) * 4
add $t2, $s0, $t2 # t2 = &A[i+1]
lw    $t3, 0($t2) # t3 = A[i + 1]
sll    $t4, $t1, 2 # t4 = i * 4
add $t4, $s0, $t4 # t4 = &A[i]
sw   $t3 , 0($t4)  # A[i] = A[i+1]
add $t1, $t1, 1 # increment the loop counter by 1
j  loop   # repeat the loop
exit:
sw $t, 20($s0)  #  A[5] = t
```