EE 180 Homework 1

Schuyler Anne Tilney-Volk, Sean William Konz

TOTAL POINTS

115 / 115

QUESTION 1

1 Problem 1 **24 / 24**

QUESTION 2

Problem 2 20 pts

2.1 a) **14 / 14**

2.2 b) **6 / 6**

QUESTION 3

Problem 3 16 pts

3.1 a) **8 / 8**

3.2 b) **8 / 8**

QUESTION 4

Problem 4 40 pts

4.1 a) **8 / 8**

4.2 b) **12 / 12**

4.3 c) **8 / 8**

4.4 d) **12 / 12**

QUESTION 5

Problem 5 15 pts

5.1 a) **10 / 10**

5.2 b) **5 / 5**

EE180 – HW1

Sean Konz

1)

Psuedo	MIPS
① move $s1, $s2	→ add $s1, $s2, $zero
② beq $s3, big, L	→ lui $at, upper(bis) add $at, $at, lower(bis) beq $s3, $at, L
③ li $s1, small	→ addi $s1, $zero, small
④ la $s2, addr	→ lui $at, upper(addr) addi $s2, $at, lower(addr)
⑤ bgt $s3, $s4, L	→ slt $at, $s4, $s3 bnq $at, $zero, L
⑥ ble $s3, $s4, L	→ slt $at, $s4, $s3 beq $at, $zero, L
⑦ addi $s1, $s2, big	→ li $s1, upper(bis) ori $s1, $s1, lower(bis) add $s1, $s1, $s2
⑧ not $s1, $s2	→ nor $s1, $s2, $s2
⑨ div $s0, $s1, $s2	→ div $s1, $s2 mflo $s0

⑩ lw $s1, big($s2)	→ lui $at, upper(bis)
	add $at, $s2, $at
	lw $s1, lower(bis)($at)

⑪ nop	→ add $zero, $zero, $zero
⑫ sgt $s0, $t1, $t2	→ slt $s0, $t2, $t1

1 Problem 1 **24 / 24**

(2)

a) lines 1, 2, and 5 load a value from the A array at index i into reg $s2. Lines 3 and 4 compute the address of the j^{th} element of the B array. Lines 6 and 7 stores elemt i+4 in the A array into memory. Line 8 adds A[i] and A[i+4] together. Line 9 gets the address of the B[j+1] element. Line 10 Stores A[i] + A[i+4] in B[j+1]

so: equivalent:

B[j+1] = A[i] + A[i+4];

b) 1) sll $t0, $s0, 2
2) add $t0, $t0, $s6
3) lw $t1, 0($t0)
4) lw $t0, 16($t0)
5) add $t0, $t0, $t1
6) sll $t1, $s1, 2
7) add $t1, $t1, $s7
8) sw $t0, 4($t1)

2.1 a) **14 / 14**

(2)

a) lines 1, 2, and 5 load a value from the A array at index i into reg $s2. Lines 3 and 4 compute the address of the j^{th} element of the B array. Lines 6 and 7 stores elemt i+4 in the A array into memory. Line 8 adds A[i] and A[i+4] together. Line 9 gets the address of the B[j+1] element. Line 10 Stores A[i] + A[i+4] in B[j+1]

so: equivalent:

B[j+1] = A[i] + A[i+4];

b) 1) sll $t0, $s0, 2
2) add $t0, $t0, $s6
3) lw $t1, 0($t0)
4) lw $t0, 16($t0)
5) add $t0, $t0, $t1
6) sll $t1, $s1, 2
7) add $t1, $t1, $s7
8) sw $t0, 4($t1)

2.2 b) **6 / 6**

(3)

a)
```
Load B subtractor
Clear accumulator
Test:
    beq subtractor, zero, exit
body:
    Add B accumulator
    decrement subtractor
    branch test
exit:
    Store Result, accumulator
```

better

```
Load B subtractor
Load zero accumulator
J test
body:
    Add B accumulator
    decrement subtractor
test:
    bne subtractor, zero, body
```

b) yes it is possible. In order to do this, you must store the value in the accumulator after each operation. For instance, we would need to add B to the accumulator, store the value into memory, then load the last index value, into the accumulator, then check if the accumulated value is greater than or equal to the reference value, then branch. The code would be:

```
Load zero accumulator
Store B
Load B
Store I
J test

body:
  Load R
  Add B
  Store R
  Load I
  decrement accumulator
  Store I

I = memory address for index
R = memory address for result
test:
    I
    bne accumulator, zero, body
```

3.1 a) **8 / 8**

(3)

a)
```
Load B subtractor
Clear accumulator
Test:
    beq subtractor, zero, exit
body:
    Add B accumulator
    decrement subtractor
    branch test
exit:
    Store Result, accumulator
```

better

```
Load B subtractor
Load zero accumulator
J test
body:
    Add B accumulator
    decrement subtractor
test:
    bne subtractor, zero, body
```

b) yes it is possible. In order to do this, you must store the value in the accumulator after each operation. For instance, we would need to add B to the accumulator, store the value into memory, then load the last index value, into the accumulator, then check if the accumulated value is greater than or equal to the reference value, then branch. The code would be:

```
Load zero accumulator
Store B
Load B
Store I
J test

body:
  Load R
  Add B
  Store R
  Load I
  decrement accumulator
  Store I

I = memory address for index
R = memory address for result
test:
    I
    bne accumulator, zero, body
```

3.2 b) **8 / 8**

(4)

a) This code iterates through the array and finds the second least frequent element in the array. It returns the value of the element in V0, and the frequency of the element in V1

b) 8 instructions before entering any loop section (before outer).
Worst case occurs when all the numbers in the array are the same except the first one (element located at index 0). This forces the outer loop to loop 300 times.

Outer loop: (13 instr) · 300 iter = 3900 instructions

Inner loop: Called 300 times for 300 iters of the outer loop
The inner loop runs 7 commands when the values being compared are the same, and 6 commands when the values being compared are different. For 299 inner loop calls, the compare value will be the most frequent element. For these 299 inner loop calls, the inner loop will loop 300 times, 299 with 7 instructions, and once with 6 instructions. For the 300th call of the inner loop, the compare value will be the second most frequent element, and will have 299 iterations with 6 instructions, and one iteration with 7 instructions. In total:

$299(299 \cdot 7 + 6) + (299 \cdot 6 + 7) = 629402$ instructions

Total: $8 + 3900 + 629402 = 633310$ instructions

4.1 a) **8 / 8**

(4)

a) This code iterates through the array and finds the second least frequent element in the array. It returns the value of the element in V0, and the frequency of the element in V1

b) 8 instructions before entering any loop section (before outer).
Worst case occurs when all the numbers in the array are the same except the first one (element located at index 0). This forces the outer loop to loop 300 times.

Outer loop: (13 instr) · 300 iter = 3900 instructions

Inner loop: Called 300 times for 300 iters of the outer loop
The inner loop runs 7 commands when the values being compared are the same, and 6 commands when the values being compared are different. For 299 inner loop calls, the compare value will be the most frequent element. For these 299 inner loop calls, the inner loop will loop 300 times, 299 with 7 instructions, and once with 6 instructions. For the 300th call of the inner loop, the compare value will be the second most frequent element, and will have 299 iterations with 6 instructions, and one iteration with 7 instructions. In total:

$299(299 \cdot 7 + 6) + (299 \cdot 6 + 7) = 629402$ instructions

Total: $8 + 3900 + 629402 = 633310$ instructions

4.2 b) **12 / 12**

c) The inner loop determines the execution time since it consists of $\frac{629402}{633310} = 99.4\%$ of the instructions in the program

for 7 instructions, the inner loop has: $1+2+3+2+2+2+2 = 14$ cycles

for 6 instructions, the inner loop has: $1+2+3+2+2+2 = 12$ cycles

$\therefore$ Execution Time: $\frac{299(299 \cdot 14) + 12 + 299(12) + 14}{900 \times 10^6} = \frac{1255228}{900 \times 10^6}$

$= .00139469\overline{8}s = 1.4ms$

---

d) cycle time increased by 25%, so

cycle time $= \frac{1}{900 \times 10^6}$ $\rightarrow$ new cycle time: $\frac{1}{900 \times 10^6} \cdot 1.25 = 1.389 \times 10^{-9} s$

Similar to (c) the inner loop significantly dominates the execution time.

The original 7 instruction case now takes: $1+2+3+2+2+2 = 12$ cycles

the original 6 instruction case now takes: $1+2+3+2+2 = 10$ cycles

$\therefore$ Execution Time: $(299(299 \cdot 12) + 10 + 299(10) + 12)(1.38 \times 10^{-9} s)$

$= .0014846s = 1.485ms$

---

the modified MIPS program is 6.1% slower than the orisinal program.

4.3 c) **8 / 8**

c) The inner loop determines the execution time since it consists of $\frac{629402}{633310} = 99.4\%$ of the instructions in the program

for 7 instructions, the inner loop has: $1+2+3+2+2+2+2 = 14$ cycles

for 6 instructions, the inner loop has: $1+2+3+2+2+2 = 12$ cycles

$\therefore$ Execution Time: $\frac{299(299 \cdot 14) + 12 + 299(12) + 14}{900 \times 10^6} = \frac{1255228}{900 \times 10^6}$

$= .00139469\overline{8}s = 1.4ms$

---

d) cycle time increased by 25%, so

cycle time $= \frac{1}{900 \times 10^6}$ $\rightarrow$ new cycle time: $\frac{1}{900 \times 10^6} \cdot 1.25 = 1.389 \times 10^{-9} s$

Similar to (c) the inner loop significantly dominates the execution time.

The original 7 instruction case now takes: $1+2+3+2+2+2 = 12$ cycles

the original 6 instruction case now takes: $1+2+3+2+2 = 10$ cycles

$\therefore$ Execution Time: $(299(299 \cdot 12) + 10 + 299(10) + 12)(1.38 \times 10^{-9} s)$

$= .0014846s = 1.485ms$

---

the modified MIPS program is 6.1% slower than the orisinal program.

4.4 d) **12 / 12**

5) Per instruction: CPI

Arith: 1 LW/SW: 10 Branch: 5

Proc: 1

Arith: 2.56 Billion LW/SW: 1.28 billion Branch: 256 million

$$\text{CPU Time: } \frac{IC \times CPI}{\text{Clock Rate}} = \frac{(2.56\times10^9 \text{ cyc}) + \left(10\frac{\text{cyc}}{\text{inst}} \cdot 1.28\times10^9 \text{ inst}\right) + \left(5\frac{\text{cyc}}{\text{inst}} \cdot .256\times10^9 \text{ inst}\right)}{4\times10^9 \frac{\text{cyc}}{s}}$$

$$= 4.165 \text{ s}$$

P=2 divide # of Arithmetic and load/store instruction by .8(2) = 1.6

CPU Time: 2.725s relative speedup: 1.53

P=4 divide # of Arithmetic and load/stores by .8(4) = 3.2

CPU Time: 1.525s relative speedup: 2.74

P=8 divide # of Arithmetic and load/stores by .8(8) = 6.4

CPU Time: .925s Relative speedup: 4.52

b) $$.00152_s = \frac{(2.56\times10^6 \text{ cyc}) + X(1.28\times10^6 \text{ inst}) + \left(5\frac{\text{cyc}}{\text{inst}} \cdot .256\times10^6 \text{ inst}\right)}{4\times10^9 \frac{\text{cyc}}{s}}$$

$\rightarrow \boxed{X = 1.75}$ the load/store CPI

5.1 a) **10 / 10**

5) Per instruction: CPI

Arith: 1 LW/SW: 10 Branch: 5

Proc: 1

Arith: 2.56 Billion LW/SW: 1.28 billion Branch: 256 million

$$\text{CPU Time: } \frac{IC \times CPI}{\text{Clock Rate}} = \frac{(2.56\times10^9 \text{ cyc}) + \left(10\frac{\text{cyc}}{\text{inst}} \cdot 1.28\times10^9 \text{ inst}\right) + \left(5\frac{\text{cyc}}{\text{inst}} \cdot .256\times10^9 \text{ inst}\right)}{4\times10^9 \frac{\text{cyc}}{s}}$$

$$= 4.165 \text{ s}$$

P=2 divide # of Arithmetic and load/store instruction by .8(2) = 1.6

CPU Time: 2.725s relative speedup: 1.53

P=4 divide # of Arithmetic and load/stores by .8(4) = 3.2

CPU Time: 1.525s relative speedup: 2.74

P=8 divide # of Arithmetic and load/stores by .8(8) = 6.4

CPU Time: .925s Relative speedup: 4.52

b) $$.00152_s = \frac{(2.56\times10^6 \text{ cyc}) + X(1.28\times10^6 \text{ inst}) + \left(5\frac{\text{cyc}}{\text{inst}} \cdot .256\times10^6 \text{ inst}\right)}{4\times10^9 \frac{\text{cyc}}{s}}$$

$\rightarrow \boxed{X = 1.75}$ the load/store CPI

5.2 b) **5 / 5**