CS3021 Computer Architecture II - Tutorial 3

Student # 16327446 - Brandon Dooley

1) RISC-I Translation

[t3.asm] - min - no optimization (RISC-I)

```
1
2   ; function to calculate min(a, b, c)
3   ;             a -> r26
4   ;             b -> r27
5   ;             c -> r28
6   ;
7   ; returns result in r1
8   ;
9   ;-------------------------------------------------------------------------
10  ;  NO OPTIMIZATION POSSIBLE
11  ;-------------------------------------------------------------------------
12
13        add r0, #4, r9                ; r9 -> g = 4
14
15  min:
16        add r26, r0, r1               ; v = a (r26)
17        sub r27, r1, r0               ; if (b < v)
18        jge min1                      ;
19        xor r0, r0, r0                ; nop in delay slot
20        add r27, r0, r1               ; v = b
21  min1:
22        sub r28, r1, r0               ; if (c < v)
23        jge min2
24        xor r0, r0, r0                ; nop in delay slot
25        add r28, r0, r1               ; v = c
26  min2:
27        ret r25, 0                    ; return(0)
28        xor r0, r0, r0                ; delay slot
29
```

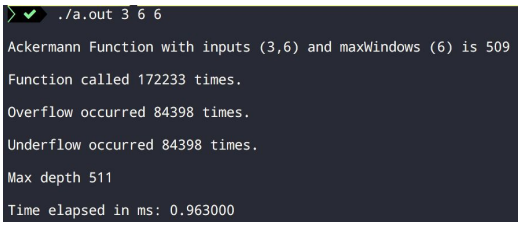
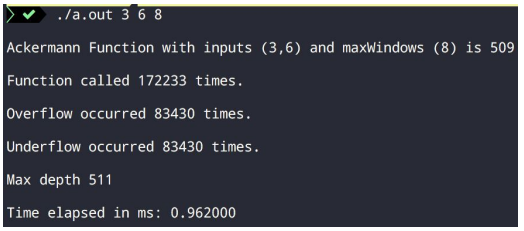
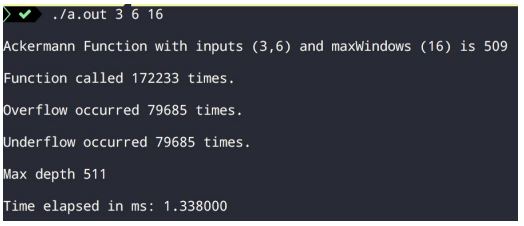
[t3.asm] - p (RISC-I)

```
30  ; function to calculate p(i, j, k, l)
31  ;             i -> r26
32  ;             j -> r27
33  ;             k -> r28
34  ;             l -> r29
35  ;
36  ; returns min(min(g, i, j), k, l) in r1
37  ;
38  ;-------------------------------------------------------------------------
39  ;  UNOPTIMISED
40  ;-------------------------------------------------------------------------
41
42  p:
43        add r9, r0, r10               ; r10 = g
44        add r26, r0, r11              ; r11 = i
45        add r27, r0, r12              ; r12 = j
46        callr r25, min                ; min(g, i, j)
47        xor r0, r0, r0                ; reset nop delay
48        add r1, r0, r10               ; r10 = min(g, i, j)
49        add r28, r0, r11              ; r11 = k
50        add r29, r0, r12              ; r12 = l
51        callr r25, min                ; min(min(g, i, j), k, l)
52        xor r0, r0, r0                ; nop reset
53        ret r25, 0                    ; return(0)
54        xor r0, r0, r0                ; nop reset
55
56  ;-------------------------------------------------------------------------
57  ;  OPTIMISED
58  ;-------------------------------------------------------------------------
59
60  p:
61        add r9, r0, r10               ; r10 = g
62        add r26, r0, r11              ; r11 = i
63        callr r25, min                ; min(g, i, j)
64        add r27, r0, r12              ; r12 = j
65        add r1, r0, r10               ; r10 = min(g, i, j)
66        add r28, r0, r11              ; r11 = k
67        callr r25, min                ; min(min(g, i, j), k, l)
68        add r29, r0, r12              ; r12 = l
69        ret r25, 0                    ; return(0)
70        xor r0, r0, r0                ; nop reset
71
```

[t3.asm] - gcd (RISC-I)

```
77  ;
78  ;-------------------------------------------------------------------------
79  ;  UNOPTIMISED
80  ;-------------------------------------------------------------------------
81
82  gcd:
83        xor r1, r1, r1                ; r1 = 0
84        sub r26, r1, r0               ; if(b==0)
85        je gcd_retA
86
87        add r26, r0, r10              ; r10 = a
88        add r27, r0, r11              ; r11 = b
89        callr r25, mod                ; mod(a, b)
90        xor r0, r0, r0                ; nop reset
91        add r27, r0, r10              ; r10 = b
92        add r1, r0, r11               ; r11 = mod(a,b)
93        callr r25, gcd                ; gcd(b, (a%b))
94        xor r0, r0, r0                ; nop result
95        ret r25, 0                    ; return(0)
96        xor r0, r0, r0                ; nop reset
97
98  gcd_retA:
99        add r26, r0, r25              ; r25 = a
100       ret r25, 0                    ; return(0)
101       xor r0, r0, r0                ; nop reset
102
103 ;-------------------------------------------------------------------------
104 ;  OPTIMISED
105 ;-------------------------------------------------------------------------
106
107 gcd:
108       xor r1, r1, r1                ; r1 = 0
109       sub r26, r1, r0               ; if(b==0)
110       je gcd_retA
111
112       add r26, r0, r10              ; r10 = a
113       callr r25, mod                ; mod(a, b)
114       add r27, r0, r11              ; r11 = b
115
116       add r27, r0, r10              ; r10 = b
117       add r1, r0, r11               ; r11 = mod(a,b)
118       callr r25, gcd                ; gcd(b, (a%b))
119       add r1, r0, r11               ; r11 = mod(a,b)
120
121       ret r25, 0                    ; return(0)
122       xor r0, r0, r0                ; nop reset
123
124 gcd_retA:
125       add r26, r0, r25              ; r25 = a
126       ret r25, 0                    ; return(0)
127       xor r0, r0, r0                ; nop reset
128
```

2) Ackermann Function

➢ Ackermann (3,6) with *maxWindows* = 6 ○ Function calls = 172233 ○ Overflows = 84398 ○ Underflows = 84398 ○ Max Depth = 511 ○ Result = 509	
➢ Ackermann (3,6) with *maxWindows* = 8 ○ Function calls = 172233 ○ Overflows = 83430 ○ Underflows = 83430 ○ Max Depth = 511 ○ Result = 509	
➢ Ackermann (3,6) with *maxWindows* = 16 ○ Function calls = 172233 ○ Overflows = 79685 ○ Underflows = 79685 ○ Max Depth = 511 ○ Result = 509	

3) Ackermann Function - Timing

The following times are calculated using the following C code on a MacBook Pro with a 2 GHz Intel Core i5:

```
// Clock time structs
clock_t startTime, endTime;

// Start clock
startTime = clock();

// Run Ackermann(3,6)
int res = 0;
res += ackermann(3, 6);

// Stop clock
endTime = clock();

// Calculate elapsed time
double elapsed = (double)(endTime - startTime) * 1000.0 / CLOCKS_PER_SEC;
```

Function Call	Time (ms)
Ackermann(3,6) with *maxWindows* = 6	0.957
Ackermann(3,6) with *maxWindows* = 8	0.959
Ackermann(3,6) with *maxWindows* = 16	0.958

** The above times are averages taken over 10 seperate function calls*

From the above results it is quite difficult to draw any significant conclusions with regards to performance differences of the Ackermann function based on the number of processor register sets.

These times would be increased or decreased depending on the underlying processor of the machine upon which the code is being executed on. They would also significantly be influenced by the clock speed of the processor.