|  |  |  |  |
| --- | --- | --- | --- |
| Object file header |  |  |  |
|  | Name | Procedure A |  |
|  | Text size | 0x140 |  |
|  | Data size | 0x40 |  |
| Text segment | Address | Instruction |  |
|  | 0 | LDR R0, [R3, #0] |  |
|  | 4 | ORR R1, R0, #0 |  |
|  | 8 | BL 0 |  |
| Data Segment | 0 | (X) |  |
| Relocation Info | Address | Instruction type | Dependency type |
|  | 0 | LDR | X |
|  | 4 | ORR | X |
|  | 8 | BL | B |
| Symbol table | Label | Address |  |
|  | X | - |  |
|  | B | - |  |
| Object file header |  |  |  |
|  | Name | Procedure B |  |
|  | Text size | 0x300 |  |
|  | Data size | 0x50 |  |
| Text segment | Address | Instruction |  |
|  | 0 | STR R0, [R3, #0] |  |
|  | 4 | B 0 |  |
|  | … | … |  |
|  | 0X180 | MOV PC, LR |  |
|  | … | … |  |
| Data Segment | 0 | (Y) |  |
|  | … | … |  |
| Relocation Info | Address | Instruction type | Dependency |
|  | 0 | STR | Y |
|  | 4 | B | FOO |
| Symbol Table | Label | Address |  |
|  | Y | -- |  |
|  | F00 | 0x180 |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Executable file header |  |  |
|  | Text Size | 0x440 |
|  | Data size | 0x90 |
| Text segment | Address | Instruction |
|  | 0040 0000 | LDR R0, 0 (r3) |
|  | 0040 0004 | ORR R1, R0, #0 |
|  | 0040 0008 | BL 0000 0130 |
|  | …. | …. |
|  | 0040 0140 | STR R0, 0000 0040(r3) |
|  | 0040 0144 | BL 0000 0174 |
|  | …. | …. |
|  | 0040 02c0 | MOV PC, LR |
|  | …. | …. |
| Data Segment | Address | Instruction |
|  | 1000 0000 | X |
|  | …. | …. |
|  | 1000 0040 | Y |
|  | …. | …. |
|  |  |  |

R3= 0x 1000 0000

0040 0140- (0040 0008+8)=0000 0130

B 0—0040 02c0- (0040 0144+8)= 0000 0174

Problem 2:

2.41.1:

Given the CPIs above, total cycles per program will be C=1\*500\*10^6+10\*300\*10^6+3\*100\*10^6=3800\*10^6

Assuming the clock cycle time is 1

The total CPU execution time is 3800\*10^6

After reducing the number of arithmetic instructions. The total cycles per program will be C=1\*500\*10^6\*0.75+10\*300\*10^6+3\*100\*10^6=375\*10^6+3000\*10^6+300\*10^6=3675\*10^6 cycles

The clock cycle time is now 1.1

So the total CPU execution time is 4042.5\*10^6

Therefore it is not a good choice

2.41.2

So the performance of arithmetic instructions is doubled—(execution time for Arithmetic instructions now)/(execution time for arithmetic instructions before)=1/2—still assuming the clock cycle time is 1, then the execution time for arithmetic instructions now is equal to 1\*500\*10^6\*1\*0.5=250\*10^6, and the total execution time for CPU is now 250\*10^6+3000\*10^6+300\*10^6=3550\*10^6, so the speedup is equal to 3800/35500=1.0704

If we improve the performance of arithmetic instructions by 10 times

The total execution time for CPU is now 1\*500\*10^6\*0.1+3000\*10^6+300\*10^6=3350\*10^6

So the speedup is 3800/3350=1.1343

Problem3

2.42.1:

CPI for arithmetic instruction=2 clock cycles per instruction

CPI for load/store instruction=6 clock cycles per instruction

CPI for branch instruction= 3 clock cycles per instruction

And 70% of the executed instructions are arithmetic instructions, 10% are load/store and 20% are branch

In this case, the overall effective CPI=2\*70%+6\*10%+3\*20%=1.4+0.6+0.6=2.6

2.42.2

Because the 25% improvement is solely in arithmetic instructions:

Let’s assume the number of cycles arithmetic instructions would take in order to achieve a 25% performance improvement is x

X\*0.7+6\*0.1+3\*0.2=2.6\*0.75=1.95

X\*0.7=0.75

X=1.07143

2.42.3

Because the 50% improvement is solely in arithmetic instructions:

Let’s assume the number of cycles arithmetic instructions would take in order to achieve a 50% performance improvement is x

X\*0.7+6\*0.1+3\*0.2=2.6\*0.5=1.3

X\*0.7=0.1

X=0.1429

Problem4

a) functTail(int x, int a) {

if (x <= 0) return a;

else if (x & 0x1) {

return functTail(x – 1, a + x);

} else {

return functTail(x – 3, a + 3 – x);

}

}

The call funct(val) now should look like functTail(val, 0);

b)

.text:  
funcTail:  
MOV r3, #0x1  
  
Loop:  
  
CMP r0, #0  
  
BLE Exit  
  
AND r4, r3, r0  
  
CMP r4, #1  
  
BNE B2  
  
ADD r1, r1, r0          @ a = a + x  
  
SUB r0, r0, #1          @ x = x - 1  
  
BL Loop  
  
Exit:  
  
@mov r0, r1  
@mov r1, r0  
@mov r0, #0 @I tried to print the result (stored in r1) out but failed  
@swi 0x6b  
  
SWI 0x11  
  
  
   
  
B2:  
  
 ADD r1, r1, #3          @ a = a + 3  
  
 SUB r1, r1, r0          @ a = a + 3 - x  
  
 SUB r0, r0, #3          @ x = x - 3  
  
 BL Loop  
  
main:  
MOV r0, #3  
MOV r1, #0  
BL funcTail  
  
SWI 0x11  
.end