**Problem 1**: Assume that Procedure A has a text size of 0x140, data size of 0x40 and Procedure B has a text size of 0x300 and data size of 0x50. Also assume r3 = 0x1000 0000, and the memory allocation strategy as shown in Figure 2.13 (Chapter 02\_COD 4e ARM.pdf). Link the object files above to form the executable file, by merging A with B (A goes first). Give explicitly the addresses for instructions and data in the executable and update the symbol table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Procedure A | | | | Procedure B | | | |
| Text Segment | Address | Instruction |  | Text Segment | Address | Instruction |  |
| 0 | LDR r0, [r3, #0] |  | 0 | STR r0, [r3, #0] |  |
| 4 | ORR r1, r0, #0 |  | 4 | B 0 |  |
| 8 | BL 0 |  | … | … |  |
| … | … |  | 0x180 | MOV pc, lr |  |
|  |  |  | … | … |  |
| Data Segment | 0 | (X) |  | Data Segment | 0 | (Y) |  |
| … | … |  | … | … |  |
| Relocation Info | Address | Instruction Type | Dependency | Relocation Info | Address | Instruction Type | Dependency |
| 0 | LDR | X | 0 | STR | Y |
| 4 | ORR | X | 4 | B | FOO |
| 8 | BL | B |  |  |  |
| Symbol Table | Address | Symbol |  | Symbol Table | Address | Symbol |  |
| … | X |  | … | Y |  |
| … | B |  | 0x180 | FOO |  |

**Answer 1:** Total text size = 0x440. Total data size = 0x90.

|  |  |  |
| --- | --- | --- |
| Text Segment | Address | Instruction |
|  | 0x00400000 | LDR r0, [r3, #0] |
|  | 0x00400004 | ORR r1, r0, #0x10000000 |
|  | 0x00400008 | BL 0x130 |
|  | … | … |
|  | 0x00400140 | STR r0, [r3, #0x40] |
|  | 0x00400144 | B 0x174 |
|  | … | … |
|  | 0x004002c0 | MOV pc, lr |
|  | … | … |
| Data Segment | Address | Data |
| 0x10000000 | (X) |
| … | … |
| 0x10000040 | (Y) |
| … | … |

**Problem 2:** (P&H ARM Edition) Exercise 2.41

Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions 300 million load/store instructions, 100 million branch instructions.

2.41.1 Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, while increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2.41.2 Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

**Answer 2:**

Given the CPIs above, total cycles per program will be: cycles.

Assuming 1 time unit per clock total execution time will be: time units.

2.41.1) After the reducing the number of arithmetic instructions, total cycles count will be: cycles.

Thus total execution time will be time units which is greater than the original.

So, this design choice is bad, because it leads to the increasing of the total execution time.

2.41.2) After the speeding up the arithmetic instruction twice, total cycles count will be cycles.

Which gives us speedup.

After the speeding up the arithmetic instruction by ten times, total cycles count will be cycles.

Which gives us speedup.

**Problem 3:** (P&H ARM Edition) Exercise 2.42

Assume that for a given program 70% of the executed instructions are arithmetic, 10%

are load/store, and 20% are branch.

2.42.1 Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

2.42.2 For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

2.42.3 For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

**Answer 3:**

2.42.1) Average CPI is

2.42.2) To get the CPI for arithmetic instruction that will lead to 25% performance improvement we need to solve the equation , which is

2.42.3) To get the CPI for arithmetic instruction that will lead to 50% performance improvement we need to solve the equation , which is

**Problem 4:**

a) Modify the following C code so that the recursion is tail-recursion.

b) Translate the tail recursion version into ARM assembly code.

funct(int x) {

if (x <= 0) return 0;

else if (x & 0x1) {

return x + funct(x-1);

} else {

return x - funct(x-1);

}

}

**Answer 3:**

1. We can’t easily transform subtraction of the function result to tail recursion, but let’s try to substitute the expression assuming x is even: . We got rid from the subtraction, so now we can easily modify the function:

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);

1. functTail:

SUB sp, sp, #4 ; allocate place on the stack for return address

STR lr, [sp, #0] ; store return address

CMP r0, #0 ; compare x to 0

BGT functTailRec ; if n > 0, go to recursive call

MOV r0, r1 ; else, return a

B functTailEnd ;

functTailRec:

STR r0, [sp, #0] ; store original argument into stack

TST r0, #1 ; check if the x is even

BEQ functTailEven ; if the x is even, jump forward

ADD r1, r1, r0 ; a = a + x

SUB r0, r0, #1 ; x = x - 1

BL functTail ; do the recursive call

B functTailEnd ; return

functTailEven: ; if the x is even

ADD r1, r1, #3 ; a = a + 3

SUB r1, r1, r0 ; a = a + 3 - x

SUB r0, r0, #3 ; x = x - 3

BL functTail ; do the recursive call

; return

functTailEnd:

LDR lr, [sp, #0] ; restore return address

ADD sp, sp, #4 ; pop items off the stack

MOV pc, lr ; return to the caller

**Problem Bonus:**

In the following assembly code, the main function reads integers from an array, calls subroutine “fact” to compute the factorial of each integer N, and prints the result fact(N) to the screen. You are asked to revise the code such that for each integer N, if N is larger than 10, it modifies the subroutine “fact” on-the-fly to compute 1+2 + …+N, instead of 1 x 2 x … x N.

main:

ldr r2, =myarray

mov r3, #0

mov r4, #4

LOOP: cmp r3, r4

bgt Stop

ldr r0, [r2, r3, LSL #2]

bl fact

mov r1, r0

MOV r0, #1 ; Load 1 into register r0 (stdout handle)

SWI 0x6b ; Print integer in register r1 to stdout

; print a space

mov r0, #1

ldr r1, =Space

swi 0x69 ; write string to stdout

add r3, r3, #1

b LOOP

Stop: SWI 0x11 ; Stop program execution

fact: sub sp, sp, #8

str lr, [sp, #0]

str r0, [sp, #4]

cmp r0, #1

bge L1

mov r0, #1

add sp, sp, #8

mov pc, lr

L1: sub r0, r0, #1

BL fact

mov r1, r0

ldr r0, [sp, #4]

ldr lr, [sp, #0]

add sp, sp, #8

mul r0, r1, r0

mov pc, lr

.data

myarray: .word 2, 3, 14, 5, 6

Space: .ascii " "

**Answer Bonus:**

main:

ldr r2, =myarray

mov r3, #0

mov r4, #4

LOOP: cmp r3, r4

bgt Stop

ldr r0, [r2, r3, LSL #2]

ldr r1, =changePos ; load position of modifiable code

cmp r0, #10 ; compare N and 10

ldrgt r5, =0xe0810000; ; if N > 10, load code of add instruction

ldrle r5, =0xe0000091; ; if N <= 10, load code of mul instruction

str r5, [r1] ; place code of instruction

factCall:

bl fact

mov r1, r0

MOV r0, #1 ; Load 1 into register r0 (stdout handle)

SWI 0x6b ; Print integer in register r1 to stdout

; print a space

mov r0, #1

ldr r1, =Space

swi 0x69 ; write string to stdout

add r3, r3, #1

b LOOP

Stop: SWI 0x11 ; Stop program execution

fact: sub sp, sp, #8

str lr, [sp,#0]

str r0, [sp,#4]

cmp r0,#1

bgt L1

mov r0, #1

add sp, sp, #8

mov pc, lr

L1: sub r0, r0, #1

BL fact

mov r1, r0

ldr r0, [sp, #4]

ldr lr, [sp, #0]

add sp, sp, #8

changePos: mul r0, r1, r0

mov pc, lr

.data

myarray: .word 2, 3, 14, 5, 6

Space: .ascii " "