CISC 260 Machine Organization and Assembly Language
Assignment # 1 Solution

1. [25 points] Convert the following numbers to other data representations. The binary is 8-bit, interpreted as two's complement.

Decimal	Binary	Hexadecimal
-98	1001 1110	**0x9E**
-39	1101 1001	**D9**
62	**0011 1110**	**3E**
91	0101 1011	0x **5B**
-8	**1111 1000**	0xF8

2. [25 points] ASCII code.

a) Decode the following bit sequence (expressed in hexadecimal):

0X49034353: I♥CS

b) Encode the following word to bit sequence (expressed in hexadecimal):

UDCIS: 55 44 43 49 53

3. With x = 1001 0001two and y = 0100 1010two representing two's complement signed integers, perform the following operations, showing all the work:

Answer:

Decimal value of x and y: x_{10} = -111 and y_{10} =74

a. x + y:

***Expected answer*:**

(-111) +74= -37_{10}

Actual Answer:

1001 0001
\+ 0100 1010
1101 1011_2 (-37_{10})

Overflow: No

b. x-y:

***Expected answer*:**

(-111)-74-= -185

Actual Answer:

y_{10} =74; y = 0100 1010_{two}

1's complement of y= 1011 0101

2's complement of y=

1011 0101
\+ 1
1011 0110

x-y=x+ 2's complement of y:

1001 0001
\+ 1011 0110
10100 0111

Overflow: yes

CISC 260 Machine Organization and Assembly Language
Assignment # 1 Solution

4. [30 points] Write a C program to implement the Booth algorithm for multiplication of signed integers, as discussed in class. You may assume the input a and b are small enough, i.e., only require 16-bit, so that the product c= a x b can fit into 32-bit. The following is a template for reading two integers a and b, and printing the product c = a x b.

Answer:

Booth's Algorithm:

```
#include <stdio.h>
int multBooth (int q, int m) {
        int a = 0;
        int q_neg1 = 0;
        int q_0 = 0;
        int i = 16;
        while (i > 0) {
                q_0 = q & 1;
                if (q_0 == 1 && q_neg1 == 0) // 10 {
                        a -= m;
                }
                else if (q_0 == 0 && q_neg1 == 1) // 01 {
                        a += m;
                }
                q_neg1 = q_0;
                m <<= 1;
                q >>= 1;
                i -= 1;
        } return a;
}
void main () {
        int q, m, a;
        printf ("Enter an integer:\n");
        scanf ("%d", &q);
        printf ("Enter an integer:\n");
        scanf ("%d", &m);
// the code of your subroutine multBooth is called below
        a = multBooth (q, m);
        printf ("the product = %d\n", a);
}
```

CISC 260 Machine Organization and Assembly Language
Assignment # 2 Solution; Fall 2021

1. ***[25 points] Given the following truth table, where X, Y, and Z are input and W is output, write the canonical expression and generate gate-level logical circuit (draw the wire diagram).***

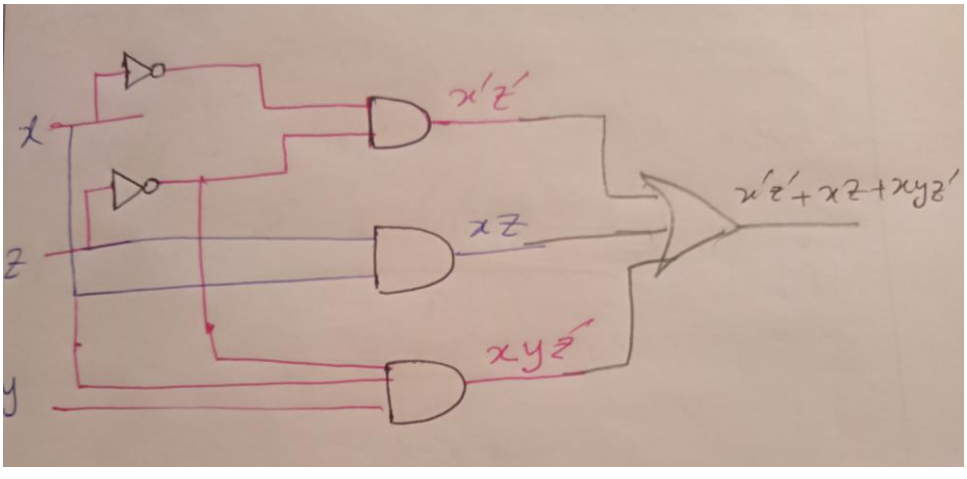
Canonical expression:

W= x'y'z'+x'yz'+xy'z+xyz'+xyz

= x'z'(y'+y)+ xz(y'+y)+xyz'

= x'z'+xz+xyz'

- ***Multiple Solution possible**



2. [25 points] Fill out the truth table for the following circuit. Note that x, y, and z are input, and F is output.

Detailed explanation:

Final output:

CISC 260 Machine Organization and Assembly Language
Assignment # 2 Solution; Fall 2021

x	y	z	F	Marks
0	0	0	1	4
0	0	1	0	3
0	1	0	0	3
0	1	1	1	3
1	0	0	1	3
1	0	1	0	3
1	1	0	0	3
1	1	1	0	3

3. [25 points] Simply the following Boolean expressions as much as you can.
a) W= y(xz' + x'z) + y'(xz' + x'z)
b) W = xy + xyz + xy'z + x'y'z
Note: x', y' and z' stand for not(x), not(y), and not(z) respectively.

Answer:

3(a): W=y(xz'+x'z)+y'(xz'+x'z)
= (xz'+x'z)(y+y')
= (xz'+x'z) *1 [(y+y')=1]
=(xz'+x'z) ***[12.5] 6.5+6***

3(b) W= xy + xyz + xy'z + x'y'z
=xy(1+z)+y'z(x+x') [(1+z)=1 and (x+x')=1]
=xy+y'z ***[12.5]: 6.5+6***

4. [25 points] You are asked to design a circuit to detect if an overflow occurs when subtracting two integers represented in two's complement: Z = X - Y. Let Sz, Sx, and Sy be the sign bit for Z, X, and Y respectively, and they are fed as input to the circuit. Let O be the output bit of the circuit, whose value is 1 if an overflow happens, and 0 if otherwise.

a) Build the truth table for O as a Boolean function of Sx, Sy, and Sz.
b) Write the canonical expression (sum-of-product) for the Boolean function defined in the part a.
c) Implement the Boolean expression defined in part b with a circuit by using AND, OR, and NOT gates. Draw the wiring diagram.

4(a) Truth Table: 10

S_x	S_y	S_z	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0

CISC 260 Machine Organization and Assembly Language
Assignment # 2 Solution; Fall 2021

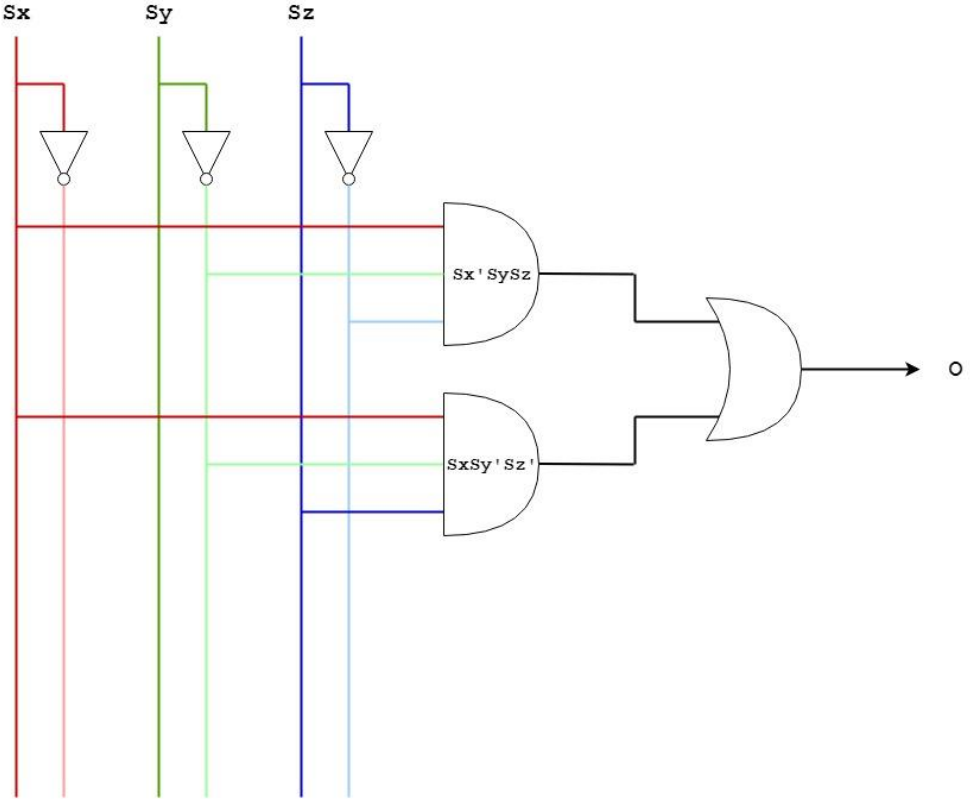
1	1	1	0

4(b) $O = S_x' S_y S_z + S_x S_y' S_z'$ **5**

CISC 260 Machine Organization and Assembly Language
Assignment # 2 Solution; Fall 2021

4 (C) Circuit Diagram ***15 for wrong equation marks figure 05, for correct equation but little mistake in figure***

10



Problem 1.
#count the number of ones in the binary representation of an integer
r0 = a, r1 = 1, r3 = 0

```
        SUB     r7, r7, r7
LOOP    BRZ     r0, EXIT
        AND     r4, r0, r1
        BRZ     r4, ELSE
        ADD     r7, r7, r1
ELSE:   SRL     r0, r0
        BRZ     r3, LOOP
EXIT:   HALT
```

```
00 0001 1 111 111 111
00 1001 0 000  001110
00 0111 1 000 001 100
00 1001 0 100  001010
00 0000 1 111 001 111
00 0011 1 000 000 000
00 1001 0 011 000010
00 1111 0 000 000 000
```

```
0000 0111 1111 1111      0x07FF
0010 0100 0000 1110      0x240E
0001 1110 0000 1100      0x1E0C
0010 0101 0000 1010      0x250A
0000 0011 1100 1111      0x03CF
0000 1110 0000 0000      0x0E00
0010 0100 1100 0010      0x24C2
0011 1100 0000 0000      0x3C00
```

Problem 2:
Image of the register files before running the program of Problem 1:

```
R0: 0111 1010
R1: 0000 0001
R2: 0000 0000
R3: 0000 0000
R4: 0000 0000
R5: 0000 0000
R6: 0000 0000
R7: 0000 0010
```

Image of the register files after running the program of Problem 1:

```
R0: 0000 0000
R1: 0000 0001
R2: 0000 0000
R3: 0000 0000
```

```
R4: 0000 0000
R5: 0000 0000
R6: 0000 0000
R7: 0000 0101
```

Problem 3. GCD

```
while (a != b) {
 if (a > b) {
   a = a - b
 } else {
   b = b -a
 }
}
return a
```

assembly code:

```
a -> r0
b -> r1
                 sub r7, r7, r7
LOOP:
                 sub r2, r1, r0
                 brz r2, exit
                 blez r2, RESET_A
                 add r1, r2, r7
                 jump LOOP
RESET_A: sub r0, r0, r1
                 jump    LOOP
exit:            add r7, r7, r0
```

Machine code:

```
00 0001 1 111 111 111
00 0001 1 001 000 010
00 1001 0 010 010 000
00 1010 0 010 001 100
00 0000 1 001 010 111
00 1110 0 000 000 010
00 0001 1 000 001 000
00 1110 0 000 000 010
00 0000 1 111 000 111
```

CISC 260 Machine Organization and Assembly Language

Assignment # 4 (Solution) Fall 2021

1. **Convert the following ARM assembly code into machine language. (Answer must be in Hexa)**

Assembly	**Hexa**
ADD R3, R1, R2 :	E0813002
LDR R10, [R5, #6] :	E595A006
SUB R3, R6, #0x24 :	E2463024
LSL R5, R4, #12 :	E1A05604

2. **<u>Three part of this question</u>**

I. Convert the following program from machine language into ARM assembly language.
Ii. reverse engineer a high-level program that would compile into this assembly language routine and write it.
Iii. Explain in words what the program does.

<u>Answer:</u>

Hexa	**ARM Assembly**

0x00008104 0xE3A0201F
0x00008108 0xE1A03230
0x0000810C 0xE2033001
0x00008110 0xE4C13001
0x00008114 0xE2522001
0x00008118 0x5AFFFFFA
0x0000811C 0xE1A0F00E
	mov r2, #0x1f
	L1: lsr r3, r0, r2
	and r3, r3, #1
	strb r3, [r1], #1
	subs r2, r2, #1
	bpl L1
	mov pc, lr

CISC 260 Machine Organization and Assembly Language

Assignment # 4 (Solution) Fall 2021

3. **Convert the following branch instructions into ARM machine code. Instruction addresses are given to the left of each instruction.**

(a) 0x0000A000 BNE LOOP
0x0000A004 ...
0x0000A008 ...
0x0000A00C
0x0000A010 LOOP ...

Answer:

BNE LOOP =
cond = 0001
op = 10
1L = 10
imm24 = decimal 2 = 0000 0000 0000 0000 0000 0010
putting all together, in hex it is 0x 1A00 0002

(b) 0x00801000 BLE DONE
... ...
0x0080204C DONE ...

Answer:

cond = 1101
op = 10
1L = 10
imm24 = 0000 0000 0000 0100 0001 0001
putting all together, in hex it is 0x DA000411

(c) 0x00104000 BL FUNC
... ...
0x0011148C FUNC ...

Answer:

cond =1110
op = 10
1L = 11
imm24 = decimal 13601 = 0000 0000 0011 0101 0010 0001
putting all together, in hex it is 0x EB003521

(d) 0x00008000 L1 ...
... ...
0x0000E00C B L1

Answer:

cond = 1110
op = 10
1L = 10
imm24 = decimal -6149 = 1111 1111 1110 0111 1111 1011
putting all together, in hex it is 0x EAFFE7FB

---

4. Implement the following C code snippets in ARM assembly language.

(i) Assume (signed) integer variables g and h are in registers R0 and R1, respectively.

```
if (g < h)
h = h + 1;
else
h = h * 2;
```

Solution 1:	**Solution 2:**

CMP R0, R1
BGE ELSE
ADD R1, R1, #1
B END
ELSE: LSL R1, R1, #2
END: HALT
	CMP R0, R1
	BLT ELSE
	LSL R1, R1, #2
	B END
	ELSE: ADD R1, R1, #1
	END: HALT

(ii) Assume R1 holds i and that R0 holds the base address of the vals array. Use as few instructions as possible.

```
int i;
int vals[200];
for (i=0; i < 200; i=i+1)
vals[i] = i;
```

CISC 260 Machine Organization and Assembly Language

Assignment # 4 (Solution) Fall 2021

Solution

MOV R1, #0
LOOP:
STR R1, [R0, R1 LSL #2]
ADD R1, R1, #1
CMP R1, #200
BLT LOOP

CISC 260 Machine Organization and Assembly Language

Assignment # 4 (Solution) Fall 2021

5. Implement functions in assembly language. You are asked to write a program in ARM assembly language to compute the TFibonacci numbers, using recursive function calls.

Answer: Multiple solution possible

```
Main: MOV  r0,#6  @ calculating Tfib(6). Main required for the assignment, only for testing
      BL   TFib
      swi  0x11
TFib:
      SUB  SP,SP,#16
      STR  LR,[SP,#0]
      CMP  r0,#2
      BGT  ELSE
      LDR  LR,[SP,#0]
      ADD  SP,SP,#16
      MOV  PC,LR
ELSE:
      STR  r0,[SP,#4]
      SUB  r0,r0,#3
      BL   TFib
      STR  r0,[SP,#8]
      LDR  r0,[SP,#4]
      SUB  r0,r0,#2
      BL   TFib
      STR  r0,[SP, #12]
      LDR  r0,[SP, #4]
      SUB  r0,r0,#1
      BL   TFib
      LDR  r2,[SP, #12]
      LDR  r1,[SP,#8]
      ADD  r0,r0,r1
      ADD  r0,r0,r2
      LDR  LR, [sp, #0]
      Add  SP,SP,#16
      MOV  PC,LR
```

Problem 1 [15 pts]. Show the IEEE 754 binary representation for the following fraction numbers in single precision. Give your answer in hexadecimal. State if each number can be represented exactly.

a. – 11.4375

b. 260.0

Answer:

a. ***– 11.4375***

Step 1: Convert it into binary: 1011.0111

Step 2: Normalize the binary number: 1.0110111 * 2^3

Step 3: Calculate the biased exponent

For single precision, bias is 2^(8-1)-1=2^7-1=127

Exponent: 3+127=130=10000010

Step 4: Store number in single precision format

Sign bit= 1 (as - sign)

Exponent= 10000010

Significand= 01101110000000000000000

Hexa =C1370000

Sign bit (1 bit)	**Exponent (8 bit)**	**Significand (23 bit)**
1	10000010	01101110000000000000000

b. ***260.0***

Step 1: Convert it into binary: 100000100.0

Step 2: Normalize the binary number: 1.00000100 * 2^8

Step 3: Calculate the biased exponent

For single precision, bias is 2^(8-1)-1=2^7-1=127

Exponent: 8+127=135=10000111

Step 4: Store number in single precision format

Sign bit= 0(as + sign)

Exponent= 10000111

Significand= 00000100000000000000000

01000011100000100000000000000000= 43820000

Problem 2 [15 pts]. What decimal number does the following bit pattern represent if it is a single precision floating-point number using the IEEE 754 standard?

a. 0xC0120000

b. 0x0DE00000

Answer

a. ***0xC0120000***

1. Convert Hexa to Binary: 1 100 0000 0 001 0010 0000000000000000

2. Separate sign bit, biased exponent, & significand:

Sign bit= 1

Biased exponent= 100 0000 0

Significand= 001 0010 0000000000000000

3.Convert biased exponent into a true exponent by subtracting bias

Biased exponent= 100 0000 0=128

128-127=1

4.Write no. as a normalized binary number

1. 001 0010 * 2^1

5.Convert it to a denormalized binary number

10.01 0010

6. Convert de-normalized binary to decimal

10.01 0010=**- 2.28125**

b. ***0x0DE00000***

1. Convert Hexa to Binary: 0 00011011 11000000000000000000000

2. Separate sign bit, biased exponent, & significand:

Sign bit= 0

Biased exponent= **000**11011

Significand=11000000000000000000000

3.Convert biased exponent into a true exponent by subtracting bias

Biased exponent= **000**11011=27

00011011- 01111111=27-127=-100

4.Write no. as a normalized binary number

1.00011011 * 2^-100

5.Convert it to a denormalized binary number

0.00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000**1000**11011

6. Convert de-normalized binary to decimal

1.38050658414E-30

Problem 3 [20 pts]. A single precision IEEE 754 number is stored in memory at address X. Write a sequence of ARM instructions to multiply the number at X by 32 and store the result back at X. You must accomplish this without using any floating-point instructions (you may ignore overflow or underflow). Note: this is a paper-pencil

Answer:

The basic idea is to deal with the exponent. You are not allowed to use any floating-point instructions;

Therefore, you can add 5 to the exponent of the number making it multiplied by 32 = 2^5.

Solution 1:

```
ldr r0, =X
ldr r1, [r0]

ldr r2, =0x02800000
add r1, r1, r2

str r1, [r0]
```

Solution 2:

```
ldr r0, =X
ldr r1, [r0]

move r2, r1
lsl r2, r2, #1
lsr r2, r2, #24

add r2, r2, #5

lsl r2, r2, #23

ldr r3, =0x80400000
and r1, r1, r3
orr r1, r1, r2

str r1, [r0]
```

Problem 4 [50 points]: Multiple solutions possible
Answer:

```
Insert:
        sub sp, sp, #8
        str lr, [sp, #0]
        str r0, [sp, #4]
        ldr r2, [r0, #0]
        ldr r3, [r1, #0]
        cmp r3, r2
        blt less_than
        str r0, [r1, #4]
        str r1, [r0, #0]
        b end
less_than:
        ldr r4, [r0, #4]
        ldr r5, [r0, #8]
        cmp r4, #0
        beq left_child
        cmp r5, #0
        beq right_child
        mov r0, r4
        bl Insert
        mov r6, r0
        ldr r0, [sp, #4]
        str r6, [r0, #4]
        b end
left_child:
        str r1, [r0, #4]
        b end
right_child:
        str r1, [r0, #8]
        b end
end:
        ldr lr, [sp, #0]
        add sp, sp, #8
        mov pc, r14
```

```
deleteMax:
        ldr r1, [r0, #0]        @ this is the max
rec:    ldr r2, [r0, #4]        @ pointer to left child
        ldr r3, [r0, #8]        @ pointer to right child
        cmp r2, #0
        bne L2                  @ non-empty left child
        cmp r3, #0
        bne goR                         @ non-empty right child

both:   ldr r2, =0xffffffff     @ both children empty
     str r2, [r0, #0]    @ clear the data and put a special value to indicate the heap is empty

        mov r0, r1              @ return the max
        mov pc, lr

L2:     ldr r4, [r2, #0]        @ left child data
        cmp r3, #0
        beq goL
        ldr r5, [r3, #0]        @ right child data
        cmp r4, r5
        bgt goL
goR:    ldr r5, [r3, #0]
        str r5, [r0, #0]

        @ check for case both children empty
        ldr r4, [r3, #4]
        ldr r5, [r3, #8]
        cmp r4, #0
        bne downR
        cmp r5, #0
        bne downR
        @ both children empty, cut the node
        mov r4, #0
        str r4, [r0, #8]
        mov r0, r1
        mov pc, lr
        @ End optional

downR:      mov r0, r3
        b rec
```

```
goL:    str r4, [r0, #0]

        @ check for case both children empty
        ldr r4, [r2, #4]
        ldr r5, [r2, #8]
        cmp r4, #0
        bne downL
        cmp r5, #0
        bne downL
        @ both children empty, cut the node
        mov r4, #0
        str r4, [r0, #4]
        mov r0, r1
        mov pc, lr
        @ End optional

downL:          mov r0, r2
        b rec
```

Problem 1 [15 points] Assume for a given processor the CPI of arithmetic instructions is 2, the CPI of load/store instructions is 6, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 600 million arithmetic instructions 250 million load/store instructions, 350 million branch instructions.

a) 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?

Answer:

Original time to run program:
CPU time = (2*600,000,000 + 6*250,000,000 + 3*350,000,000) * CC = 3,750 million CC
New time to run program:
CPU time = (2*0.75 * 600 million + 6*250 million + 3*350 million) * 1.1CC = 3,795 million CC

No, this is not a good design choice, as it will increase the CPU time it will take to run the program.

---

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

Answer:

Original CPU time (from part a): 3,750 million CPU time

Doubling performance of arithmetic instructions:
(2/2*600 million) + 6*250 million + 3 * 350 million = 3,150 million CPU time
Speedup= 3,750 million / 3,150 million = 1.19 (approx)

Improving arithmetic instruction performance by 10x:
(2/10 * 600 million) + 6*250 million + 3 * 350 million = 2,670 million CPU time
Speedup = 3,750 million / 2,670 million = 1.40 (approx)

Problem 2 [15 points] Assume that for a given program, 60% of the executed instructions are arithmetic, 15% are load/store, and 25% are branch.

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

Answer:

Average CPI = (.6 * 2 + .15*5 + .25 * 3) = 2.7 CPI

---

b) For a 15% improvement in performance, if load/store and branch instructions are not improved at all, how many cycles, on average, may an arithmetic instruction take?

Answer:

CPI_old / CPI_new = 1.15
CPI_old = 2.7
CPI_new = 0.6 * x + 0.15 * 5 + 0.25 * 3
2.7 = 1.15 * (.6 * x + .15 *5 + .25 * 3)
x = 1.413

Acceptable answer (not exactly correct)
(.85 * 2.7) = .6 * x + .15 *5 + .25*3
2.295 = .6x + .75 + .75
.795 = .6x
X = 1.325
In order to achieve a 15% improvement in performance by only adjusting the CPI of arithmetic instructions, the new arithmetic instructions should take 1.325 CPI

---

c) For a 15% improvement in performance, what is the speedup needed for branch instructions, if the other type of instructions remain unchanged?

Answer:

CPI_old / CPI_new = 1.15
CPI_old = 2.7
CPI_new = 0.6 * 2 + 0.15 * 5 + 0.25 * x
2.7 = 1.15 * (.6 * 2 + .15 * 5 + .25 * x)
x = 1.59
Speedup = 3/1.59 = 1.87

Acceptable answer (not exactly correct)
(.85 *2.7) = .6*2 + .15*5 + .25*x
2.295 = 1.2 + .75 + .25x
.345 = .25x
X = 1.38
Speedup = 3 / 1.38 = 2.17 = Speedup of 2.17 required for branch instructions to see a 15% improvement in performance if other instructions remain unchanged

---

Problem 3 ***[15 points] Modify the following C code so that the recursion is tail-recursion.***

```
Fib (n) {
if (n == 0 || n == 1)
 return n;
else
return Fib(n-2) + Fib(n-1); }
```

Explain briefly why a tail recursion is more efficient than the original version.

Answer:

Code: multiple solution possible

```
tail_ fib(n, a , b ){
   if n == 0:
      return a
   if n == 1:
      return b
   return tail_fib(n - 1, b, a + b);
}
 int main(){
  int n,a,b,X;
n = 9;
a=0;
b=1;
X=tail_fib(n, a, b);
printf(“”%d”, X)
}
```

Reason:

There is no task remaining after the recursive call in tail recursion. As a result, optimizing the code is simple for the compiler. There is no need to store the address of the called function in the stack while using tail recursion.

Problem 4. [25 points] Consider the following ARM assembly code to compute the sum of square of elements in an integer array. Pointer to the array is in r0, and the size of the array is in r1

```
add r1, r0, r1 LSL #2
mov r3, #0
1. Loop: ldr r2, [r0, #0]
2. mul r2, r2, r2
3. add r3, r3, r2
4. add r0, r0, #4
5. cmp r0, r1
6. bne Loop
```

a) [5 pts] Compute the number of cycles needed for each loop iteration in a 5 stage pipelined ARM processor, with one delayed cycle for ldr, 3 delayed cycles for mul if the results of ldr and mul are used in the next instruction.

Answer:

Here, *the results of ldr and mul are used once in the next instruction.*
Number of line inside the loop=6 (1 delay each instruction)
Number of ldr=1 (1 delay each)
Number of mul=1 (3 delay each)
6+1+3= 10 cycles

---

b) [8 pts] Unroll the loop to compute two elements in each iteration. What is the number of cycles per iteration now?

Answer:

Here, *the results of ldr and mul are used twice in the next instruction.*

```
add r1, r0, r1 LSL #2
mov r3, #0
Loop:
1. ldr r2, [r0, #0]
2. mul r2, r2, r2
3. add r3, r3, r2
4. ldr r2, [r0, #4]
5. mul r2, r2, r2
6. add r3, r3, r2
7. add r0, r0, #8
8. cmp r0, r1
9. bne Loop
```

Add another set of ldr, add, and mul instruction for the second element.

Number of line inside the loop=9 (1 cycle each instruction)
Number of ldr=2 (1 delay each)
Number of mul=2 (3 delay each)
9+1+1+3+3=17 cycles

Alternative answer (if ignore the register use protocol and use r4 without saving it)
Loop:
1. ldr r2, [r0, #0]
2. mul r2, r2, r2
3. add r3, r3, r2
4. ldr r4, [r0, #4]
5. mul r4, r4, r4
6. add r3, r3, r4
7. add r0, r0, #8
8. cmp r0, r1
9. bne Loop
The number of cycles is 17.

---

c) [12 pts] Reorder the code in part b) to further reduce the number of cycles per iteration. Give the minimum number of cycles per iteration with your reordered code.

Answer:
Here, *the results of ldr and mul are used once in the next instruction.*
add r1, r0, r1 LSL #2
mov r3, #0
Loop:
1. ldr r2, [r0, #0]
2. mul r2, r2, r2
3. add r3, r3, r2
4. ldr r2, [r0, #4]
5. add r0, r0, #8
6. mul r2, r2, r2
7. cmp r0, r1
8. add r3, r3, r2
9. bne Loop

9 + 1 (for line 1) +3 (for line 2) + 0 (for line 4) + 2 (for line 6) = 15 cycles

Alternative answer (if ignore the register use protocol and use r4 without saving it)
Loop:

1. ldr r2, [r0, #0]
2. ldr r4, [r0, #4]
3. mul r2, r2, r2
4. mul r4, r4, r4
5. add r0, r0, #8
6. add r3, r3, r2
7. cmp r0, r1
8. add r3, r3, r4
9. bne Loop

9 instruction + 0 delay for line 1 (because r2 is not used in next instruction) + 0 delays for line 2 (because lines 3-5 can be executed during that) + 0 delay for line 4 (because lines 5-6 can be executed during that) = 9 cycles

---

Problem 5 [30 points] Consider the following repeating sequence of LDR addresses (given in hexadecimal):
40, 44, 48, 4C, 70, 74, 78, 7C, 80, 84, 88, 8C, 90, 94, 98, 9C, 04, 8C, 10, 14, 18, 1C, 20.
Assuming least recently used (LRU) replacement for associative caches, determine the effective miss rate if the sequence is input to a 16-word cache with the following specifications, ignoring startup effects (i.e., compulsory misses).

7C
78
74
70
20
9C 1C
98 18
94 14
90 10
4C 8C
48 88
44 84 04
40 80

(a) direct mapped cache, b = 1 word

Answer:
Miss rate = 17/23. Addresses 70-7C and 20 use unique cache blocks and are not removed once placed into the cache, therefore have benefit of temporal locality. Miss rate is 17/23.

(b) fully associative cache, b = 1 word

Answer:
Miss rate = 22/23. A repeated sequence of length greater than the cache size produces no hits for a fully-associative cache using LRU. Here the only hit is address 8C, which appear twice in the repeat sequence, with a separation less than the cache size 16.

(c) two-way set associative cache, b = 1 word

Answer:

7C 9C 1C
78 98 18
74 94 14
70 90 10
4C 8C
48 88
44 84 04
40 80 20

Miss rate = 19/23. The repeated sequence makes at least three accesses each pass to these sets: 0, 1, 4, 5, 6 and 7. Using LRU replacement, each value in those sets must be replaced each pass through, and thus cannot be benefited from temporal locality. The four addresses 48, 4C, 88, and 8C will not overwrite each other, and therefore will not suffer any miss (except for the compulsory misses).

(d) direct mapped cache, b = 2 words

Answer:

78-7C
70-74
20-24
98-9C, 18-1C
90-94, 10-14
48-4C, 88-8C
40-44, 80-84, 00-04

Since each block will be 2 words (b = 2), this means that for a 16-word cache, there must be S = C / (K * 2) = 16 / (1 * 2) = 8 sets.

Miss rate = 9/23. Data words from consecutive locations are stored in each cache block. The larger block size is advantageous since accesses in the given sequence are made primarily to consecutive word addresses. A block size of two cuts the number of block fetches in half since two words are obtained per block fetch. The address of the second word in the block will always hit in this type of scheme (e.g. address 44 of the 40-44 address pair). Thus, non-compulsory misses can only happen for sets to which more than two address-pairs are mapped; these sets are: 0, 1, 2, and 3. The first address in an address-pairs mapped to those sets will suffer a miss. The missed addresses are: 40, 80, 04, 48, 88, 90, 10, 98, and 18. Thus, the miss rate is 9/25.

Bonus Problem [30 pts]. Write in ARM assembly language a program that takes as input a string, check if it is a palindrome or not, and print your answer as "Palindrome" or "Not palindrome" correspondingly.

Answer:
Multiple solution possible

CISC 260 Machine Organization and Assembly Language

Practice Midterm Exam

This is an open-note exam. You are allowed to use notes. You are NOT allowed to use electronic devices except standard calculators.

1. [25 points] Data representations and arithmetics

a. Convert 33_{ten} into a 8-bit two's complement binary number.

Answer: 0010 0001

b. What decimal number does the following two's complement 8-bit binary number represent?

1100 1010 $= -54_{ten}$

c. Is there an overflow for an 8-bit machine when subtracting a two's complement integer x from a two's complement integer y as given below? Show your work.

x = 1000 1011 and y = 0111 0100

Answer:

X is negative and y is positive. Therefore, y-x is adding two positive integers, where overflow occurs when the result is negative.

-x = 0111 0101

0111 0100 (y)

0111 0101 (x)

1110 1001 (y-x)

Therefore, there is an overflow.

d. Show the negation of the following integer in two's complement.

X = $1101\ 0110\ 0111\ 0101_{two}$

Answer: -x = $0010\ 1001\ 1000\ 1011_{two}$

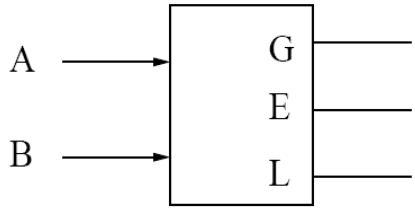
e. In multiplying the following two integers A and B , how many times the (properly shifted) multiplicand is added to the (intermediate) product C = A x B if the multiplication is implemented using the shift-add algorithm?

A = 1010 0101
B = 0110 1001

Answer: 4.

2. [20 points] Boolean Logic and Gates

A comparator circuit has two 1 bit inputs A and B and three 1 bit outputs G (greater), E (Equal) and L (less than)



G = 1, if A > B
0, otherwise

E = 1, if A = B
0, otherwise

L = 1, if A < B
0, otherwise

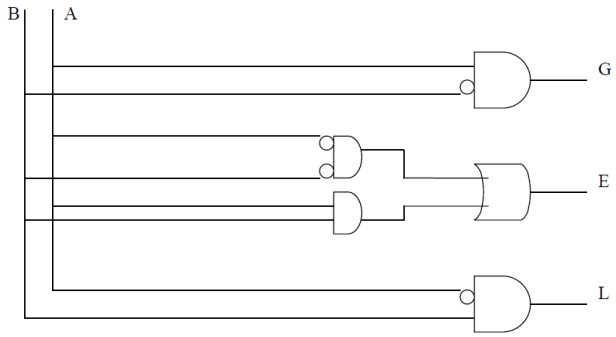
a. Fill out the truth table

A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

b. Write the Boolean expression in canonical form corresponding to the above truth table

G = A &~B; E = (~A&~B) | (A & B); L = ~A & B

c. Implement the circuit by using AND, OR and NOT gates. Draw the wiring diagram.



3. [25 points] ARM Instruction set

   a. If register r4 has a value 0x f000 000c , what is the value in r0 as the result of running the following ARM assembly language program ?

```
        CMP r4, #0
        BLE L1
        MOV r5, #1
        B L2
L1:     MOV r5, #2
L2:     MOV r0, r5
```

   Write the value in decimal: **r0** = 2

   b. For the following ARM assembly code,

```
Address                code
-------------------------------------------------------------
0x0000 1000        Main: MOV r4, #5
0x0000 1004              BL FOO
0x0000 1008              SWI 0x11
0x0000 100C         FOO: MOV r5, #1
0x0000 1010          L1: CMP r4, #0
0x0000 1014              BLE L2
0x0000 1018              MUL r6, r5, r4
0x0000 101C              MOV r5, r6
0x0000 1020              SUB r4, r4, #1
0x0000 1024              B L1
0x0000 1028         L2:  MOV r0, r5
0x0000 102C              MOV pc, r14
```

      i. When the program halts, what are the values in the following registers?
      r0 = 120
      r14 = 0x0000 1008
      r15 = 0x0000 1008

      ii. How many time has the instruction “MUL r6, r5, r4” been executed?

      5

      iii. What does the program compute?

The program computes factorial for the integer stored in r4, in this case, it is 5! = 120.

4. [30 points] ARM Assembly programming

The following is a C function that takes an integer n > 0 and returns 1 + ... + n.

```
int sum_to (int n) {
     if (n<=1) return 1;
     else
          return n + sum_to(n-1);
}
```

a) You are asked to translate the program into ARM assembly code. You may assume that n is in r0, and write the returned value in r1.

b) If n = 5, how many activation frames are pushed onto the stack during the execution of the above program.

Answer:

a)

```
sum_to: sub sp, sp, #8
      str lr, [sp,#0]
      str r0, [sp,#4]
      cmp r0,#1
      bgt else
      mov r1, #1
      add sp, sp, #8
      mov pc, lr
else: sub r0, r0, #1
      BL sum_to
      mov r2, r1
      ldr r1, [sp, #4]
      ldr lr, [sp, #0]
      add sp, sp, #8
      add r1, r2, r1
      mov pc, lr
```

b) 5

University of Delaware
Computer and Information Science
CISC 260 – A Sample Final Exam

1. This is an open notes exam. You are not allowed to use any electronic devices except standard calculators.
2. Check that you have all pages. There are 4 problems for a total of 100 points.
3. Write you name on every page in the space provided
4. If you need more space, use the back of the same page. But do not feel obligated to "fill up" all the space that is provided.
5. Give clear, concise answers to each question.

Problem	Grade
1	
2	
3	
4	
Total	

1. [30 points] Short Answers

a. Write down the IEEE 754 binary representation of the number -0.75 in single precision.
Answer:

1 01111110 10000000000000000000000

b. What decimal number does the bit pattern 1010 1101 0001 0000 0000 0000 0000 0000 represent if it is a floating point number in IEEE 754 standard?

Answer:
$-1.125_{ten} \times 2^{-37}$

c. Is there an overflow for an 8-bit machine when adding a two's complement integer x to a two's complement integer y as given below? Show your work. x = 0100 1011 and y = 0111 0100

Answer: Yes, there is an overflow, because of the negative value for sum of two positive integers.

```
0111 0100
0100 1011     (+
-----------------
1011 1111
```

d. In linking stage, does the label "else" in the following code require relocation or not?

```
BEQ  else
```

Solution: No relocation is required, because reference to label "else" is pc-relative, i.e., does not need the absolution address.

e. Is the following recursion tail recursive? If not, convert it to a tailed recursion (only at the high level code, not the assembly code.)

```
int SquareSum(int n) {
     if (n == 1) return 1;
     return n*n + SquareSum(n - 1);
}
```

Solution: No, because after recursive call there is extra computation to do.

```
int SquareSum1(int n, int q) {
     if (n == 1) {
               return q +1n;
     }
     return SquareSum1(n - 1, n*n + q);
}
```

Call SquareSum1 with q=0

2. [25 points] Below is a function that inserts a node into an linked list.

```
void insert(node** root, node* a_node) {

  if((a_node)->data <= (*root)->data) {
        (a_node)->next = *root;
     *root = a_node;
  } else if((*root)->next != NULL) {
    if ((a_node)->data <= ((*root)->next)->data){
      (a_node)->next = (*root)->next;
      (*root)->next = a_node;
    } else {
      insert(&(*root)->next, a_node);
    }
  } else {
     (*root)->next = a_node;
  }

}
```

Translate the function into ARM assembly code. You can assume that the two arguments are put in r0 and r1 respectively by the caller.

A node in the linked list is structured as follows

data	Pointer to the next node

```
Insert: @ insert a new link (pointer in r1) to root (pointer to pointer in r0)
        sub sp, sp, #8
        str lr, [sp]
        ldr r2, [r0]        @ r2 has pointer to the root
        ldr r3, [r2]        @ r3 has data in root node
        ldr r4, [r1]        @ r4 has data in new node
        cmp r4, r3
        bgt Moveback
        str r2, [r1, #4]    @ new_node -> root
        str r1, [r0]        @ update new node as root
        add sp, sp, #8
        mov pc, lr
    Moveback:
        @ check if root is the last node
        ldr r3, [r2, #4]  @ r3 has pointer to node after root
        cmp r3, #0          @ check to see if r3 is null
        bne hasMore1
        str r1, [r2, #4]  @ root -> new_node
        add sp, sp, #8
        mov pc, lr
    hasMore1:
```

```
        ldr r5, [r3] @ r5 has the data of the node after the root
        cmp r4, r5
        bgt hasMore2
        str r3, [r1, #4]
        str r1, [r2, #4]
        add sp, sp, #8
        mov pc, lr
    hasMore2:
        str r2, [sp, #4]  @ save the pointer to root
        str r3, [r0]          @ put pointer to node after root into [r0]
        BL Insert
        ldr lr, [sp]
        ldr r2, [sp, #4] @ retrieve the saved pointer to root
        str r2, [r0]         @ put it in [r0]
        add sp, sp, #8
        mov pc, lr
```

3. [20 points]

```
address                .text
0x0040 0000    main:ldr  r0,  =x
0x0040 0004         bl   foo
0x0040 0008         mov  pc, lr
0x0040 000c    foo: mov  r5, #0
0x0040 0010         cmp  r0, #0
0x0040 0014         blt  L1
0x0040 0018         add  r5, r5, #1
0x0040 001c         sub  r0, r0, #1
0x0040 0020         b    foo
0x0040 0024    L1:  mov  r0, r5
0x0040 0028         mov  pc, lr

                   .data
0x1000 0000                x: .word 10
```

For the above ARM assembly program, answer the following questions.

a) Show the address next to each instruction according to the ARM conventions for memory allocation.
Answer: see above.

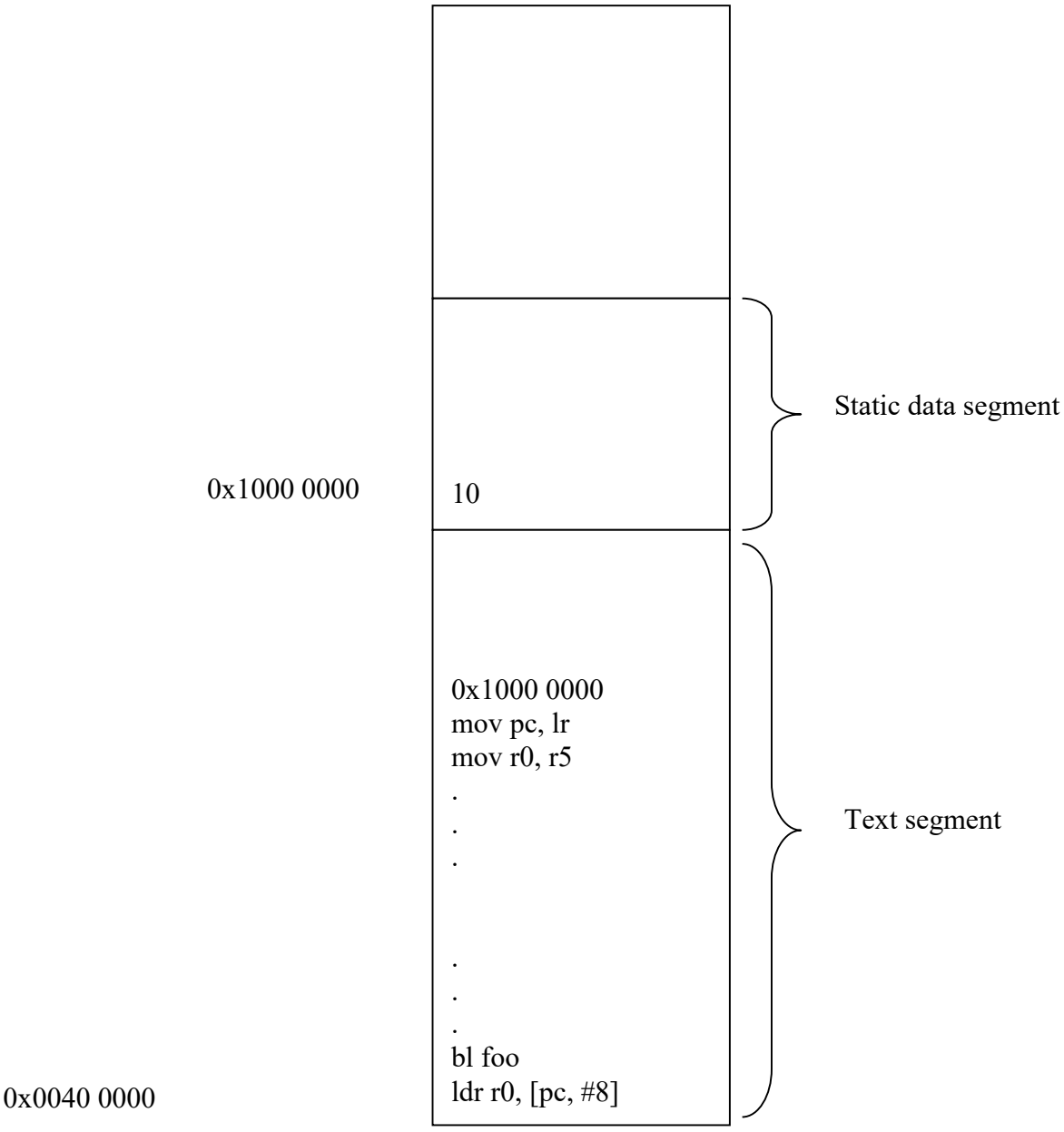
b) Draw the symbol table listing the labels and their addresses
Answer:

Symbol	address
main	0x00400000
foo	0x0040000C
L1	0x00400024
x	0x10000000

c) Convert "blt L1" into machine code (written in hex)

Answer: machine code: BA000002
See slides 6-8 in lecture notes "assembler_linker_loader.pdf"

d) Sketch a memory map showing where data and instructions are stored.


0x1000 0000
10
Static data segment
0x1000 0000
mov pc, lr
mov r0, r5
.
.
.
.
.
.
bl foo
ldr r0, [pc, #8]
Text segment
0x0040 0000

4. [25 points] Consider the following ARM assembly code

```
        mov   r3, #0
        mov   r7, #10
Loop:   ldr   r1, [r2, #0]
        mul   r1, r1, r4
        add   r2, r2, #4
        add   r3, r3, #1
        cmp   r7, r3
        bne   Loop
```

a) Compute the number of cycles needed for each loop iteration. Assume CPI is 1 for data processing, 5 for data transfer, and 2 for branching.

Answer: 5+1+1+1+1+2 = 11

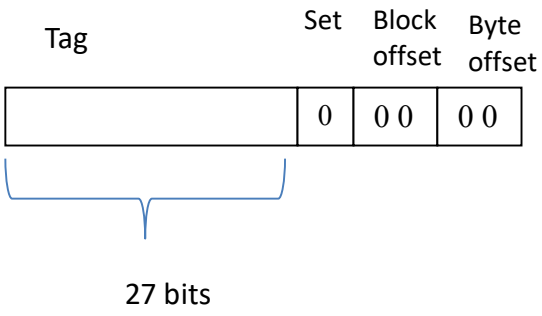
b) What is the average CPI for the above code?

Answer: total instruction count: 2 + 6 x 10 = 62
Total number of cycles = 2 + 11 x 10 = 112
Average CPI = 112/62 = 1.81

c) Assume the clock cycle is 200ps, what is the total CPU time of running the above code.

Answer: CPU time = IC x CPI x CC = 62 x 1.81 x 200ps = 22,444ps

d) What type of locality does this code have for accessing the data in memory? If the instruction **ldr** can load four adjacent words into the cache, what is the miss rate? Note that the cache has two sets, each set has a block of 4 words, with memory mapping as shown below, where the values are only placeholder to indicate the numbers of bits in each field.



Answer: spatial locality.
Miss rate = 3/10