Problem 1

An instruction is stored at location 700 with its address field at location 702. The address field has the value 120. The location 120 contains the value 10. Registers R1, R2, and R3 contains the values 140, 120 and 100 respectively. What is the effective address of the instruction operand- for each of the following addressing modes? **Show your work.**

a. Direct Addressing mode
b. Register Indirect if the location 702 contains the value 2 instead of 120
c. Register Indirect if the location 702 contains the value 3 instead of 120
d. Immediate
e. PC-relative
f. Indexed assuming that R1 is the index register.
g. Indexed assuming that R2 is the index register.
h. PC-relative if the address field is in location 701 instead of 702.

[8 marks]

Solution

a. Direct: EA = 120
b. Register Indirect : EA = address inside R2 = 120
c. Register Indirect : EA = address inside R3 = 100
d. Immediate: EA = 702.
e. PC-relative: EA = PC + value inside instruction = 703 + 120 = 823
f. Indexed: EA = content of R1 + value inside instruction = 140 + 120= 260
g. Indexed: EA = content of R2 + value inside instruction = 120 + 120= 240
h. PC-relative: EA = PC + value inside instruction = 702 + 120 = 822

Problem 2

Consider a non-pipelined machine with 5 stages of lengths 20 ns, 40 ns, 60 ns, 50 ns, 30 ns. Suppose we introduce pipelining on this machine. Assume that when introducing pipelining, the clock adds 2ns latch.

a. How much time does it take to execute 200 instructions on the non-pipelined machine?

b. How much time does it take to execute 200 instructions on the pipelined machine?

c. What is the speedup obtained from pipelining for **100 instructions**?

Show your work.

[5 marks]

Solution:

a. Execution time $_{\text{non-pipelined}} = nt_n = (20+40+60+50+30)x\ 200 = 40000$ ns

b. Execution time $_{\text{pipelined}} = (k+n-1)t_p = (5+199)x\ 62 = 12648$ ns

c. Execution time $_{\text{non-pipelined}} = nt_n = (20+40+60+50+30)x\ 100 = 20000$ ns

Execution time $_{\text{pipelined}} = (k+n-1)t_p = (5+99)x\ 62 = 6448$ ns

Speed up = Execution time $_{\text{non-pipelined}}$ / Execution time $_{\text{pipelined}} = 20000/6448 = 3.1$

Problem 3

Consider a byte addressing memory and a hypothetical 32-bit microprocessor having 32-bit instructions composed of three fields: The first upper 6 bits contains the Opcode, then 8 bits to represent an immediate operand and the remaining the second operand address. What is the maximum directly addressable memory capacity (in bytes)? **Show your work.**

[1 mark]

Solution:

Opcode 6 bits, immediate operand is 8-bits, therefore:

Second operand address = 32 – (6+8) = 18 bits

Therefore, the maximum directly addressable memory capacity (in bytes) → $2^{18} = 8$ MB

Problem 4

Write an ARM7 assembly program to evaluate the following equation (12-2) * (10-5) * (4-2). Your program should involve two subroutines, one responsible for subtracting two numbers, and one responsible for multiplication of three numbers.

The subtraction subroutine should get its two parameters via memory, and the returned result should be stored in the first memory location available after the call, whereas the multiplication subroutine should get its three parameters via memory and store its returned value in the memory location after the call.

[9 marks]

Solution:

```
        AREA Problem4, CODE, READWRITE
        ENTRY
        BL SUB1
        DCD 12
        DCD 2
NUM1 DCD 0
        BL SUB1
        DCD 10
        DCD 5
NUM2 DCD 0
        BL SUB1
        DCD 4
        DCD 2
NUM3 DCD 0
        BL MUL1
```

```
        DCD 0
        B EXT

SUB1    LDR R0, [LR]
        ADD LR, LR, #4
        LDR R1, [LR]
        ADD LR, LR, #4
        SUB R3, R0, R1
        STR R3, [LR]
        ADD LR, LR, #4
        MOV PC, LR

MUL1 LDR R1, NUM1
        LDR R2, NUM2
        LDR R3, NUM3
        MUL R4, R1, R2
        MUL R5, R3, R4
        STR R5, [LR]
        ADD LR, LR, #4
        MOV PC, LR
EXT
        END
```

Problem 5

A digital computer has a memory unit with 30 bits per cell. Each instruction on this computer is 15-bits divided as 4 bits for Opcode and 11 bits for the operand direct address part. Two instructions are packed in one memory word. The computer has the following registers: MAR (11-bits), MDR (30-bits) and IR (30-bits). Write a procedure in your own words for fetching and executing instructions for this computer.

[3 marks]

Solution:

1. Read 30-bit double instruction from memory to IR and then increment PC by 1
2. Decode opcode1
3. Put the operand address1 (11-bits) into MAR
4. Fetch the operand and store it in MDR
5. Execute instruction1
6. Decode opcode2
7. Execute instruction2 using address2
8. Put the operand address2 (11-bits) into MAR
9. Fetch the operand and store it in MDR
10. Execute instruction2
11. Go to Step1 again.

Problem 6

a) Three enhancements with the following speedups are proposed for a new machine: Speedup (a) = 30, Speedup (b) = 20, and Speedup (c) = 15. Assume that for some set of programs, the fraction of use is 25% for enhancement (a), 30% for enhancement (b), and 20% for enhancement (c). Calculate the speedup overall gained after applying the enhancements? **Show your work.**

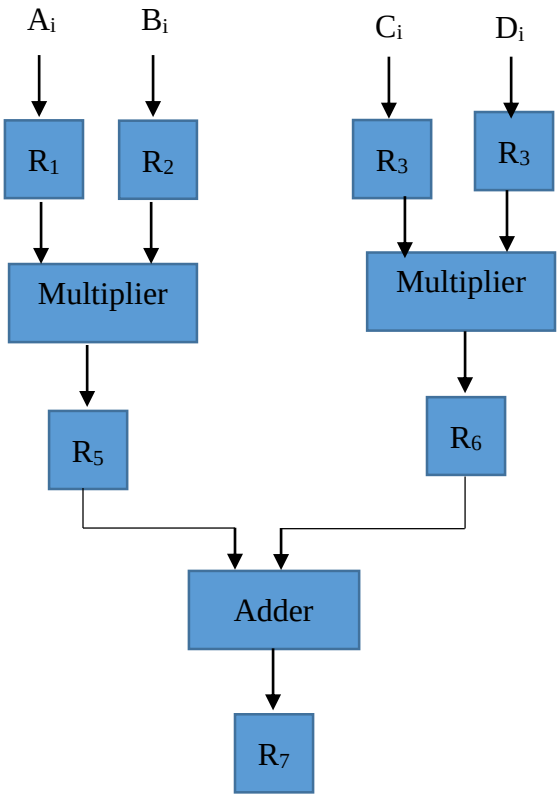
Solution:

$$SU_{a,b,c} = \frac{1}{(1 - F_a - F_b - F_c) + \frac{F_a + F_b + F_c}{SU_a + SU_b + SU_c}}$$

$$= \frac{1}{(1 - 0.25 - 0.3 - 0.2) + \frac{0.25 + 0.3 + 0.2}{30 + 20 + 15}} = 3.8$$

b) Suppose that we want to perform the combined multiply and add operations with a stream of numbers: Ai * Bi + Ci*Di for i=1,2,...,7. Each suboperation is to be implemented in a segment within a pipeline. Each segment has one or 2 registers and a combinational circuit. Specify a pipeline configuration to carry out this task. List the contents of all registers in the pipeline.

[8 marks]



Clock Pulse Number	Segment1				Segment2		Segment3
	R1	R2	R3	R4	R5	R6	R7
1	A1	B1	C1	D1	------	-----	----------
2	A2	B2	C2	D2	A1*B1	C1*D1	----------
3	A3	B3	C3	D3	A2*B2	C2*D2	A1*B1 + C1*D1
4	A4	B4	C4	D4	A3*B3	C3*D3	A2*B2 + C2*D2
5	A5	B5	C5	D5	A4*B4	C4*D4	A3*B3 + C3*D3

Problem 7

Write an ARM7 assembly program that contains a subroutine responsible for calculating the factorial of a positive number. Your subroutine should get its number via memory, and the returned result should be stored in the first memory location available after the call. The subroutine have to check whether that the number is not negative number otherwise the code should convert it to its positive value and proceed with the positive number.

[8 marks]

Solution:

```
        AREA Problem7, CODE, READWRITE
        ENTRY
        BL FACT
        DCD -4
        DCD 0
        B EXT

FACT LDR R0,[LR]
        ADD LR, LR, #4
```

```
        MOV R1, #1      ; i=1
        MOV R3, #1      ; FACTORIAL=1
        CMP R0, #0
        BGE LOOP
        MOV R5, #-1
        MUL R6, R0, R5
        MOV R0, R6
LOOP    CMP R1, R0
        BGT DONE
        MUL R4, R3, R1
        MOV R3, R4
        ADD R1, #1
        B LOOP

DONE STR R3, [LR]
        ADD LR, LR, #4
        MOV PC ,LR
EXT
        END
```

Problem 8

Write an ARM7 assembly language that given an array “Arr1” of size 8. The array contains 4 positive numbers and 4 negative numbers. Write an ARM7 assembly program that contains a subroutine that reads element by element in the array, if the number is positive multiply it by 2 using the shift operation and stores it in array “Arr2”, and if the number is negative divide it by 2 using the shift operation and stores it in array “Arr3”.

Assume arrays Arr2 and Arr3 are of size 4 and Arr1 is defined in the first line after the call, Arr2 is in the line, whereas Arr3 is in the third line after the call.

[8 marks]

Solution:

```
        AREA Problem8, CODE, READWRITE
        ENTRY
        BL Arrays
ARR1 DCD -4, 5, -10, 7, -2, 9, 2, -8
ARR2 DCD 0, 0, 0, 0
ARR3 DCD 0, 0, 0, 0
         B EXT
Arrays LDR R1, =ARR1
          LDR R2, =ARR2
          LDR R3, =ARR3

LOOP CMP R4, #8
        BEQ DONE
        LDR R5, [R1]
        ADD R1, #4

       CMP R5, #0
       BLT NEG
       MOV R6, R5,LSL #1
       STR R6, [R2]
      ADD R2, #4
```

```
        B CNT


NEG  MOV R6, R5,ASR #1
        STR R6, [R3]
        ADD R3, #4
CNT  ADD R4, #1
        B LOOP


DONE ADD LR, LR, #68
          MOV PC ,LR
EXT
      END
```