SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

ANALYTICAL QUESTIONS

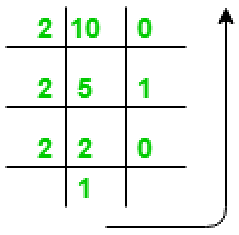
CSA12	**COMPUTER ARCHITECTURE**

<u>NUMBER SYSTEMS</u>

1. Decimal to Binary

$(10.25)_{10}$

Integer part :



$(10)_{10} = (1010)_2$

Fractional part :

$0.25 \times 2 = 0.50$
$0.50 \times 2 = 1.00$ $\downarrow$

$(0.25)_{10} = (0.01)_2$

Note: Keep multiplying the fractional part with 2 until decimal part 0.00 is obtained.

$(0.25)_{10} = (0.01)_2$

Answer: $(10.25)_{10} = (1010.01)_2$

2. Binary to Decimal

$(1010.01)_2$

$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 8+0+2+0+0+0.25 = 10.25$

$(1010.01)_2 = (10.25)_{10}$

3. Decimal to Octal

$(10.25)_{10}$

$(10)_{10} = (12)_8$

Fractional part:

0.25 x 8 = 2.00

Note: Keep multiplying the fractional part with 8 until decimal part .00 is obtained.

$(.25)_{10} = (.2)_8$

Answer: $(10.25)_{10} = (12.2)_8$

4. Octal to Decimal

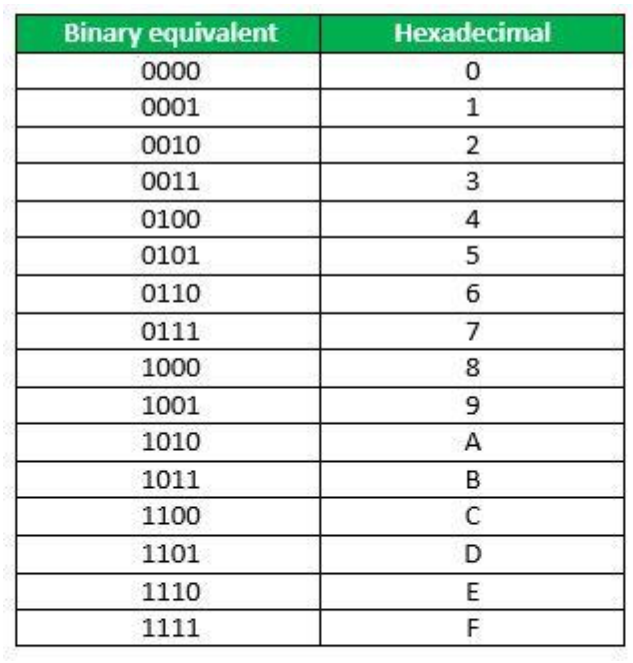
$(12.2)_8$

$1 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} = 8+2+0.25 = 10.25$

$(12.2)_8 = (10.25)_{10}$

5. Hexadecimal to Binary

To convert from Hexadecimal to Binary, write the 4-bit binary equivalent of hexadecimal.



Binary equivalent	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

$(3A)_{16} = (00111010)_2$

6. Binary to Hexadecimal

To convert from Binary to Hexadecimal, start grouping the bits in groups of 4 from the right-end and write the equivalent hexadecimal for the 4-bit binary. Add extra 0’s on the left to adjust the groups.

1111011011

$\underline{0011}$ $\underline{1101}$ $\underline{1011}$
$(001111011011)_2 = (3DB)_{16}$

7. Binary to Octal
To convert from binary to octal, start grouping the bits in groups of 3 from the right end and write the equivalent octal for the 3-bit

binary. Add 0’s on the left to adjust the groups.

Example:
111101101

$\underline{111}$ $\underline{101}$ $\underline{101}$
$(111101101)_2 = (755)_8$

8. $(101101)_2$ in decimal is

$= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 1 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$
$= 32 + 8 + 4 + 1$
$= (45)_{10}$

9. $(24)_8$ in decimal is

$= 2\times8^1+4\times8^0$
$= (20)_{10}$

10. 1 A number system uses 10 as the base. The number is 654. What is the LSB and MSB of this number?

A) 6 and 4 B) 4 and 6 C) 0 and 1 D) 10 and 01.
Answer: MSB stands for the most significant Bit and the LSB stands for the Least Significant Bit. If you take a look at the number 654, we ought to represent it in a base 10 (Decimal number system).

So we can write it as $6\times10^2 + 5\times10^1 + 4 \text{ x}10^0$. Thus the number 6 is the most significant in the sense that it contains the bulk of the value of the number. And the number 4 is the least significant bit because it contains the least bulk of the value. Hence the answer is B that is 4 and 6. We can say that in the decimal representation, the number to the left is the MSB and the number to the right is the LSB.

11. How would you represent 10111 in the decimal number system?

A) 23

B) 24

C) 25

D) 22

Ans: A) 23

12. The LSB and MSB in the following number are: 1220

A) 1 & 0

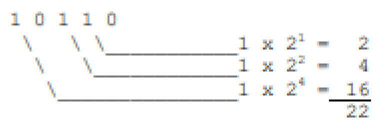
B) 0 & 1

C) 10

D) 01

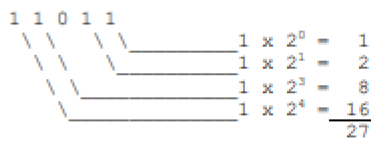
Ans: A) 0 & 1

13. convert 10110_2 to decimal.



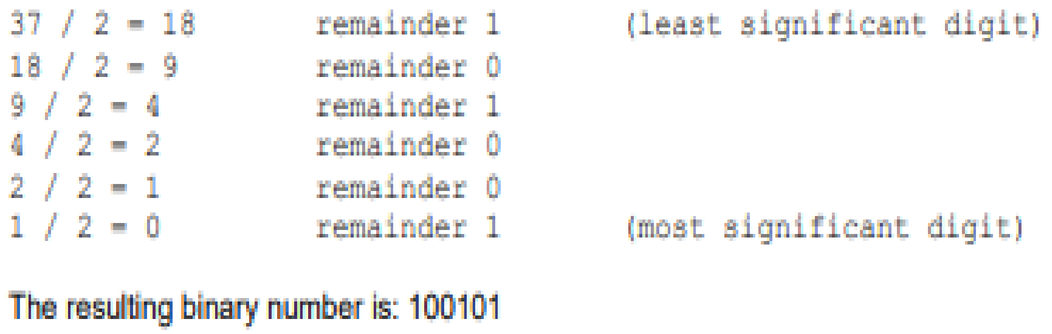
```
1 0 1 1 0
 \   \ \___________1 x 2^1 =   2
  \   \____________1 x 2^2 =   4
   \_______________1 x 2^4 =  16
                              22
```

14. convert 11011_2 to decimal



```
1 1 0 1 1
 \ \   \ \_________1 x 2^0 =   1
  \ \   \__________1 x 2^1 =   2
   \ \_____________1 x 2^3 =   8
    \______________1 x 2^4 =  16
                              27
```

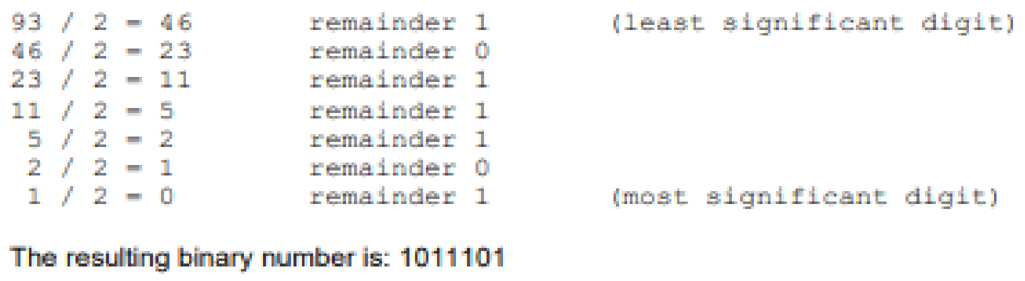
15. convert 37_{10} to binary.



```
37 / 2 = 18       remainder 1       (least significant digit)
18 / 2 = 9        remainder 0
9 / 2 = 4         remainder 1
4 / 2 = 2         remainder 0
2 / 2 = 1         remainder 0
1 / 2 = 0         remainder 1       (most significant digit)
```

The resulting binary number is: 100101

16. convert 93_{10} to binary.



```
93 / 2 = 46       remainder 1       (least significant digit)
46 / 2 = 23       remainder 0
23 / 2 = 11       remainder 1
11 / 2 = 5        remainder 1
 5 / 2 = 2        remainder 1
 2 / 2 = 1        remainder 0
 1 / 2 = 0        remainder 1       (most significant digit)
```

The resulting binary number is: 1011101

17. Convert the binary number 10110101 to a hexadecimal number

Divide into groups for 4 digits 1011 0101

Convert each group to hex digit B 5

$B5_{16}$

18. Convert the binary number 0110101110001100 to hexadecimal

Divide into groups of 4 digits 0110 1011 1000 1100

Convert each group to hex digit 6 B 8 C

$6B8C_{16}$

19. Convert the hex number 374F into binary

3 7 4 F

Convert the hex digits to binary 0011 0111 0100 1111

0011011101001111_2

20. The decimal number 136 would be represented in BCD

136 = 0001 0011 0110

1 3 6

21. The following is a 16 bit number encoded in packed BCD format:

01010110 10010011

This is converted to a decimal number as follows:

0101 0110 1001 0011

5 6 9 3

The value is 5693 decimal

22. Find the 2's complement of the following 8 bit number 00101001

11010110 First, invert the bits

+ 00000001 Then, add 1

= 11010111

The 2’s complement of 00101001 is 11010111

23. Find the 2’s complement of the following 8 bit number 10110101

01001010 Invert the bits

+ 00000001 then add 1

= 01001011

The 2’s complement of 10110101 is 01001011

24. 10101_2 to decimal

Step 1	10101_2	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

25. 12570_8 to decimal

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

26. $19FDE_{16}$ to decimal

Step	Binary Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536+ 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

27. 29_{10} to Binary

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

29_{10} to Binary Number : 11101_2

28. 11101_2 to Decimal

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Binary Number : 11101_2 = Decimal Number : 29_{10}

30. 25_8 to Binary

Step 1 - Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

25_8 = Decimal Number : 21_{10}

Step 2 - Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number : 21_{10} = Binary Number : 10101_2

Octal Number : 25_8 = Binary Number : 10101_2

31. 10101_2 to Octal

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	2_8 5_8
Step 3	10101_2	25_8

Binary Number : 10101_2 = Octal Number : 25_8

32. 10101_2 to Hexadecimal

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	1_{10} 5_{10}
Step 3	10101_2	15_{16}

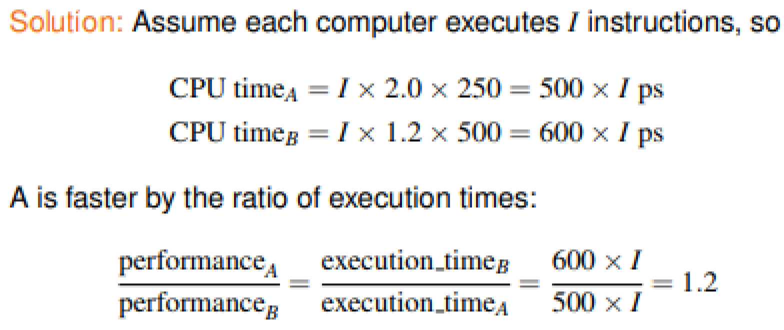
Binary Number : 10101_2 = Hexadecimal Number : 15_{16}

33. 15_{16} to Binary

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	1_{10} 5_{10}
Step 2	15_{16}	0001_2 0101_2
Step 3	15_{16}	00010101_2

Hexadecimal Number : 15_{16} = Binary Number : 10101_2

34. Computers A and B implement the same ISA. Computer A has a clock cycle time of 250 ps and an effective CPI of 2.0 for some program and computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program. Which computer is faster and by how much?



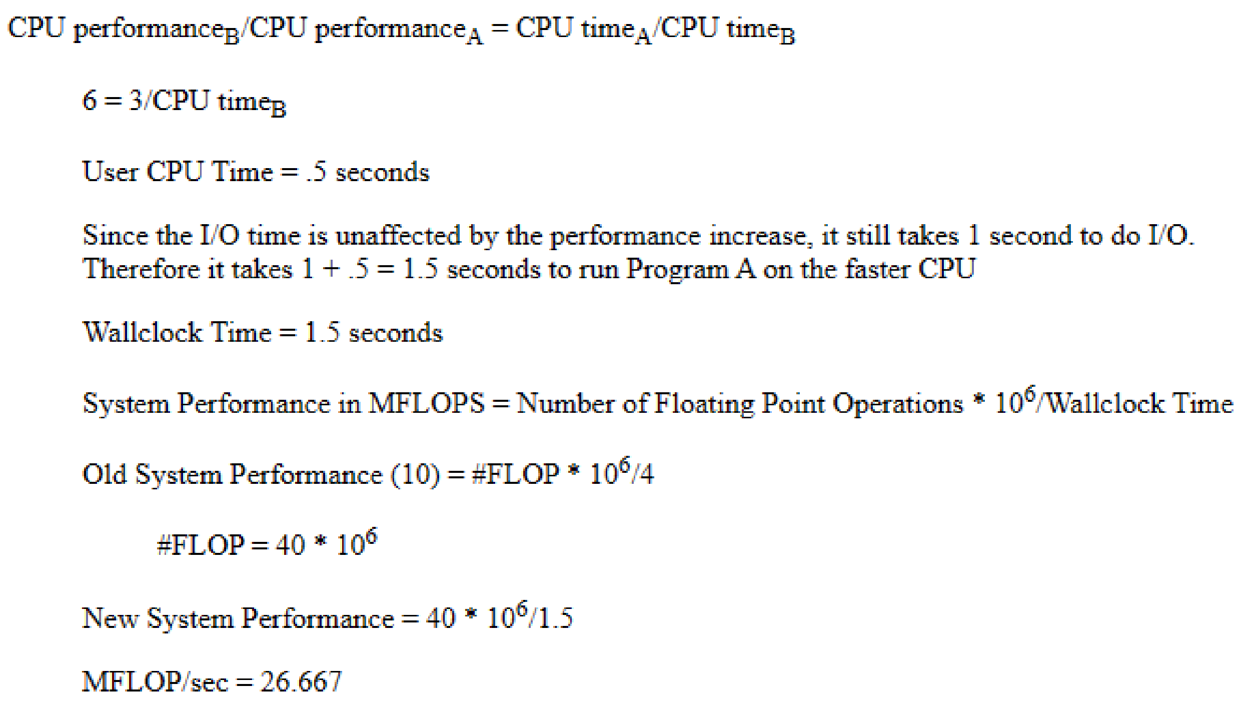
Solution: Assume each computer executes I instructions, so

$$\text{CPU time}_A = I \times 2.0 \times 250 = 500 \times I \text{ ps}$$
$$\text{CPU time}_B = I \times 1.2 \times 500 = 600 \times I \text{ ps}$$

A is faster by the ratio of execution times:

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times I}{500 \times I} = 1.2$$

35. Suppose that when Program A is run, the user CPU time is 3 seconds, the elapsed wall clock time is 4 seconds, and the system performance is 10 MFLOP/sec. Assume that there are no other processes taking any significant amount of time, and the computer is either doing calculations in the CPU, or doing I/O, but it can't do both at the same time. We now replace the processor with one that runs six times faster, but doesn't affect the I/O speed. What will the user CPU time, the wall clock time, and the MFLOP/sec performance be now?



CPU performance_B/CPU performance_A = CPU time_A/CPU time_B

6 = 3/CPU time_B

User CPU Time = .5 seconds

Since the I/O time is unaffected by the performance increase, it still takes 1 second to do I/O. Therefore it takes 1 + .5 = 1.5 seconds to run Program A on the faster CPU

Wallclock Time = 1.5 seconds

System Performance in MFLOPS = Number of Floating Point Operations * 10^6/Wallclock Time

Old System Performance (10) = #FLOP * 10^6/4

#FLOP = 40 * 10^6

New System Performance = 40 * 10^6/1.5

MFLOP/sec = 26.667

36. Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

a. Which processor has the highest performance expressed in instructions per second?

b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

c. We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

Ans:

a.. P1: 3GHz / 1.5 = 2 * 10^9 instructions per second P2: 2.5GHz / 1.0 = 2.5 * 10^9 instructions per second P3: 4GHz / 2.2 = 1.82 * 10^9 instructions per second So P2 has the highest performance among the three.

b. Cycles: P1: 3GHz * 10 = 3 * 10^10 cycles P2: 2.5GHz * 10 = 2.5 * 10^10 cycles P3: 4GHz * 10 = 4 * 10^10 cycles Num of instructions: P1: 3GHz * 10 / 1.5 = 2 * 10^10 instructions P2: 2.5GHz * 10 / 1.0 = 2.5 * 10^10 instructions P3: 4GHz * 10 / 2.2 = 1.82 * 10^10 instructions

c. Execution time = (Num of instructions * CPI) / (Clock rate) So if we want to reduce the execution time by 30%, and CPI increases by 20%, we have: Execution time * 0.7 = (Num of instructions * CPI * 1.2) / (New Clock rate) New Clock rate = Clock rate * 1.2 / 0.7 = 1.71 * Clock rate New Clock rate for each processor: P1: 3GHz * 1.71 = 5.13 GHz P2: 2.5GHz * 1.71 = 4.27 GHz P3: 4GHz * 1.71 = 6.84 GH

37. You are on the design team for a new processor. The clock of the processor runs at 200 MHz. The following table gives instruction frequencies for Benchmark B, as well as how many cycles the instructions take, for the different classes of instructions. For this problem, we assume that (unlike many of today's computers) the processor only executes one instruction at a time.

Instruction Type	Frequency	Cycles
Loads & Stores	30%	6 cycles
Arithmetic Instructions	50%	4 cycles
All Others	20%	3 cycles

- Calculate the CPI for Benchmark B.

If we say that there are 100 instructions, then:

30 of them will be loads and stores.

50 of them will be arithmetic instructions.

20 of them will be all others.

(30 * 6) + (50 * 4) + (20 * 3) = 440 cycles/100 instructions

Therefore, there are 4.4 Cycles per instruction.

38. The CPU execution time on the benchmark is exactly 11 seconds. What is the ``native MIPS" processor speed for the benchmark in millions of instructions per second?

The formula for calculating MIPS is:

MIPS = Clock rate/(CPI * 10^6)

The clock rate is 200MHz so...

MIPS = (200 * 10^6)/(4.4 * 10^6) = 45.454545

39. The hardware expert says that if you double the number of registers, the cycle time must be increased by 20%. What would the new clock speed be (in MHz)?

Clock time = 1/Cycle Time

Cycle Time = 1/Clock Time

Cycle Time = 1/(200 * 10^6) = 5 * 10^{-9}

The cycle time is then increased by 20%:

(5 * 10^{-9}) * 1.2 = 6 * 10^{-9}

The new clock rate is thus:

1/(6 * 10^{-9}) = 166.667 * 10^6 or 166.667 MHz

40. The compiler expert says that if you double the number of registers, then the compiler will generate code that requires only half the number of Loads & Stores. What would the new CPI be on the benchmark?

    There were 100 instructions in part b, so we will reduce the number of loads and stores by

    half, and this will reduce the total number of instructions. So the new instruction mix will be:

    15 Loads and Stores

    50 Arithmetic Instructions

    20 All Others

    The total number of instructions is now 85, so the answer is:

    ((15 * 6) + (50 * 4) + (20 * 3)) / 85 = 350 cycles/ 85 instructions = 4.12 CPI

41. How many CPU seconds will the benchmark take if we double the number of registers (taking into account both changes described above)?

    CPU seconds = (Number of instructions * Number of Clocks per instructions)/Clock Rate

    First thing we need to do, is calculate the number of instructions which execute in 11 seconds on the new benchmark - the one with half the number of loads and stores.

    To do this, we will need to figure out how many instructions execute on the original benchmark in 11 seconds. Since we know the MIPS or how many *Millions of Instructions Per Second* for the original benchmark, we say:

    $(45.45 * 10^6) * 11 = 500 * 10^6$ instructions in 11 seconds

    Now we need to figure out how many of those are Loads and Stores so:

$(500 * 10^6) * .3 = 150 * 10^6$ are Load and Store instructions because the chart says that 30% of all instructions are Loads and Stores. Now we need to cut this number in half, because the new benchmark says that we have half the number of loads and stores , but the cycle time increases by 20%. Therefore there are only $75 * 10^6$ loads and stores. This also means that there are now less total instructions, $425 * 10^6$ total instructions.

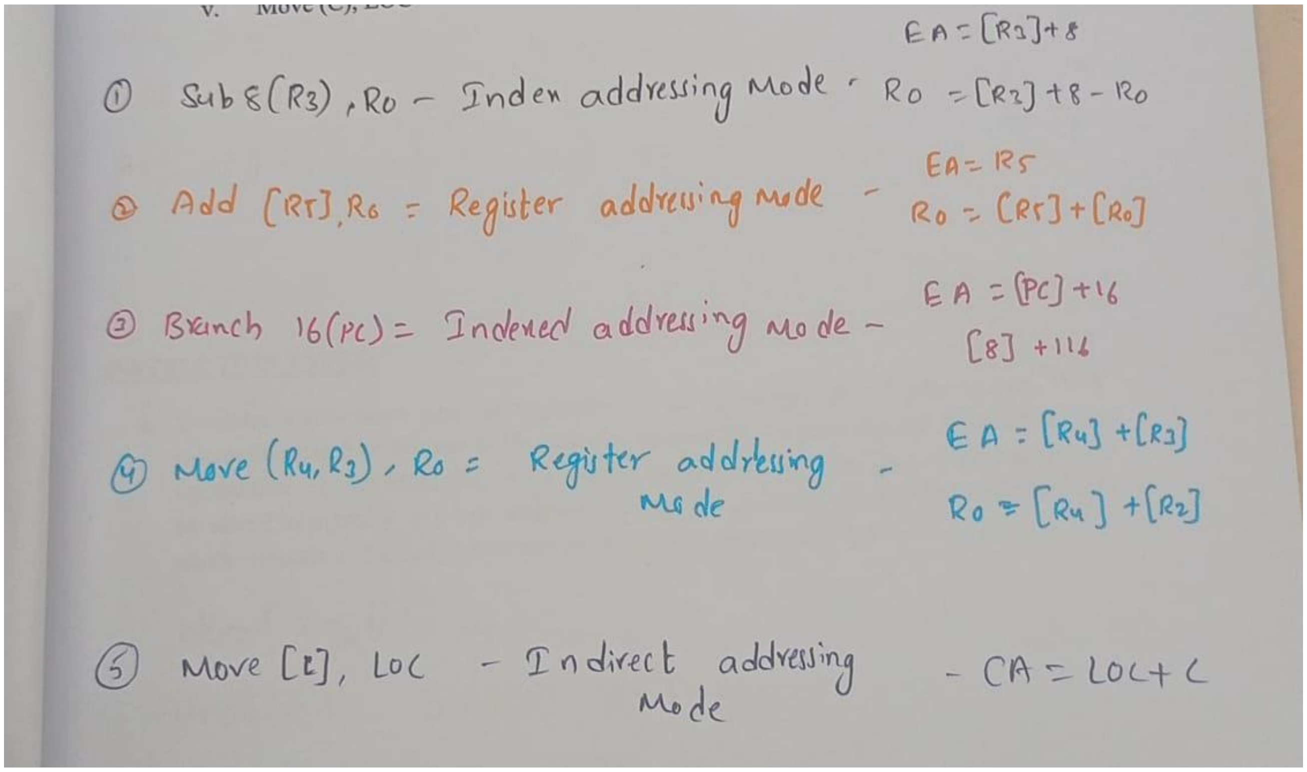
The final solution is:

$((425 * 10^6) * 4.12)/(166.667 * 10^6) = 10.548$ seconds

Day 2

ADDRESSING MODES:

1. Find the addressing mode used to indicate the location of the source operands and also calculate the effective address in the following instructions
    i. Sub 8(R3), R0
    ii. Add (R5), R0
    iii. Branch 16(PC)
    iv. Move (R4,R3), R0
    v. Move (C), LOC



① Sub 8(R3), Ro – Inden addressing Mode · EA = [R3]+8
Ro = [R3] + 8 - Ro

② Add [R5], R0 = Register addressing mode – EA = R5
R0 = [R5] + [R0]

③ Branch 16(PC) = Indexed addressing mode – EA = [PC] + 16
[8] + 116

④ Move (R4, R3), Ro = Register addressing mode – EA = [R4] + [R3]
Ro = [R4] + [R2]

⑤ Move [C], LOC – Indirect addressing Mode – CA = LOC + C

2. Write a sequence of instructions that will compute the value of *y = x2 + 2x + 3* for a given x using

    a. three-address instructions
    b. two-address instructions

c. one-address instructions

Solution

a. three-address instructions

```
b.
c.      Mult z, x, x
d.      Mult y, 2, x
e.      Add y, y, z
f.      Add y, y, 3
```

g. two-address instructions

```
h.
i.      Move z, x
j.      Mult z, x
k.      Move y, 3
l.      Add y, z
m.      Move z, x
n.      Mult z, 2
o.      Add y, z
```

p. one-address instructions

```
q.
r.      Move x
s.      Mult x
t.      Store z
u.      Move x
v.      Mult 2
w.      Add z
x.      Add 3
y.      Store y
```

3. Write procedures for reading from and writing to a FIFO queue, using a two-address format, in comjunction with:

a. indirect addressing
b. relative addressing

Solution

```
a.
b.      Reading: Loadindirect R3, R1
c.               Inc R1
```

```
d.      Writing: Storeindirect R2, Data
e.               Inc R2
```

f.

g. if we have a fixed base register called BR:

```
h.
i.      Reading: Move temp, BR
j.               Move BR, R1
k.               Loadrel R4, offset
l.               Inc R1
m.               Move BR, temp
n.      Writing: Move temp, BR
o.               Move BR, R2
p.               Storerel offset, Data
q.               Inc R2
r.               Move BR, temp
```

4. Write a sequence of instructions that will compute $\sum_{i=1}^{100}$aixi where *A* = [a1, a2, ..., a100] and *X* = [x1, x2, ..., x100] represent arrays that are stored in the main memory. Use two-address instructions, in conjunction with:

   a. direct addressing
   b. relative addressing
   c. indexed addressing with an auto-increment mode

Solution

Direct addressing:

```
1.
2.      Move Sum, 0
3.      Load Temp1, 1000
4.      Load Temp2, 2000
5.      Mult Temp1, Temp2
6.      Add Sum, Temp1
7.      ...
8.      Load Temp1, 1099
9.      Load Temp2, 2099
10.     Mult Temp1, Temp2
11.     Add Sum, Temp1
```

relative addressing

```

Move Sum, 0
Loadrel Temp1, 0
Loadrel Temp2, 100
Mult Temp1, Temp2
Add Sum, Temp1
...
Loadrel Temp1, 99
Loadrel Temp2, 199
Mult Temp1, Temp2
Add Sum, Temp1
```

indexed addressing with an auto-increment mode

```

    Move Sum, 0
    Move IR, 0
L1: Loadindex Temp1, 1000
    Loadindex Temp2, 2000
    Mult Temp1, Temp2
    Add Sum, Temp1
    Cmp IR, 100
    Bne L1
```

5. Add a dedicated base register (*BR*) to the 16-bit processor shown in Figure 9.9, and then show the changes this requires in the instruction set and the processor schematic.

Solution

Add one instruction to load the base:

```

Lbase Address : BR <-- Address
```

Change relative addressing:

```

Lrel Dest, Base : RF[Dest] <-- Mem[BR + Offset]
```

Srel Srcl, Base : Mem[BR + Offset] <-- RF[Srcl]

6. Consider a three word machine instruction -

ADD A[R_0], @B

The first operand (destination) "A[R_0]" uses indexed addressing mode with R_0 as the index register. The second operand operand (source) "@B" uses indirect addressing mode. A and B are memory addresses residing at the second and the third words, respectively. The first word of the instruction specifies the opcode, the index register designation and the source and destination addressing modes. During execution of ADD instruction, the two operands are added and stored in the destination (first operand).

The number of memory cycles needed during the execution cycle of the instruction is-

1. 3
2. 4
3. 5
4. 6

Solution-

For the first operand,

- It uses indexed addressing mode.
- Thus, one memory cycle will be needed to fetch the operand.

For the second operand,

- It uses indirect addressing mode.
- Thus, two memory cycles will be needed to fetch the operand.

After fetching the two operands,

- The operands will be added and result is stored back in the memory.
- Thus, one memory cycle will be needed to store the result.

Total number of memory cycles needed

= 1 + 2 + 1

= 4

Thus, Option (B) is correct.

7. Consider a hypothetical processor with an instruction of type LW R_1, 20(R_2), which during execution reads a 32-bit word from memory and stores it in a 32-bit register R_1. The effective address of the memory location is obtained by the addition of a constant 20 and the contents of register R_2. Which of the following best reflects the addressing mode implemented by this instruction for operand in memory?

1. Immediate Addressing
2. Register Addressing
3. Register Indirect Scaled Addressing
4. Base Indexed Addressing

Solution-

Clearly, the instruction uses base indexed addressing mode.

Thus, Option (D) is correct.

8. The memory locations 1000, 1001 and 1020 have data values 18, 1 and 16 respectively before the following program is executed.

MOVI	R_s, 1	Move immediate
LOAD	R_d, 1000(R_s)	Load from memory

ADDI	R_d, 1000	Add immediate
STOREI	0(R_d), 20	Store immediate

Which of the statements below is TRUE after the program is executed?

1. Memory location 1000 has value 20
2. Memory location 1020 has value 20
3. Memory location 1021 has value 20
4. Memory location 1001 has value 20

Solution-

Before the execution of program, the memory is-

Now, let us execute the program instructions one by one-

Instruction-01: MOVI R_s, 1

This instruction uses immediate addressing mode.

- The instruction is interpreted as Rs ← 1.
- Thus, value = 1 is moved to the register R_s.

Instruction-02: LOAD R_d, 1000(R_s)

This instruction uses displacement addressing mode.

- The instruction is interpreted as R_d ← [1000 + [R_s]].
- Value of the operand = [1000 + [R_s]] = [1000 + 1] = [1001] = 1.
- Thus, value = 1 is moved to the register R_d.

Instruction-03: ADDI R_d, 1000

This instruction uses immediate addressing mode.

- The instruction is interpreted as R_d ← [R_d] + 1000.
- Value of the operand = [R_d] + 1000 = 1 + 1000 = 1001.
- Thus, value = 1001 is moved to the register R_d.

Instruction-04: STOREI 0(R_d), 20

This instruction uses displacement addressing mode.

- The instruction is interpreted as $0 + [R_d] \leftarrow 20$.
- Value of the destination address = $0 + [R_d] = 0 + 1001 = 1001$.
- Thus, value = 20 is moved to the memory location 1001.

Thus,

- After the program execution is completed, memory location 1001 has value 20.
- Option (D) is correct.

9. Consider the following memory values and a one-address machine with an accumulator, what values do the following instructions load into accumulator?

- Word 20 contains 40
- Word 30 contains 50
- Word 40 contains 60
- Word 50 contains 70

Instructions are-

1. Load immediate 20
2. Load direct 20
3. Load indirect 20
4. Load immediate 30
5. Load direct 30
6. Load indirect 30

Solution-

Instruction-01: Load immediate 20

This instruction uses immediate addressing mode.

- The instruction is interpreted as Accumulator ← 20.
- Thus, value 20 is loaded into the accumulator.

Instruction-02: Load direct 20

This instruction uses direct addressing mode.

- The instruction is interpreted as Accumulator ← [20].
- It is given that word 20 contains 40.
- Thus, value 40 is loaded into the accumulator

Instruction-03: Load indirect 20

This instruction uses indirect addressing mode.

- The instruction is interpreted as Accumulator ← [[20]].
- It is given that word 20 contains 40 and word 40 contains 60.
- Thus, value 60 is loaded into the accumulator.

Instruction-04: Load immediate 30

This instruction uses immediate addressing mode.

- The instruction is interpreted as Accumulator ← 30.
- Thus, value 30 is loaded into the accumulator.

Instruction-02: Load direct 30

This instruction uses direct addressing mode.

- The instruction is interpreted as Accumulator ← [30].
- It is given that word 30 contains 50.
- Thus, value 50 is loaded into the accumulator

Instruction-03: Load indirect 30

This instruction uses indirect addressing mode.

- The instruction is interpreted as Accumulator ← [[30]].
- It is given that word 30 contains 50 and word 50 contains 70.
- Thus, value 70 is loaded into the accumulator.

10. At memory address 200, a two instruction LOAD AC is stored. At location 201, the address stored is 500. At location 202 next instruction is stored. The following numbers are stored at the different memory locations, as shown in this table.

Memory Location (Address)	Memory Content
399	**450**
400	**700**

500	**800**
600	**900**
702	**325**
800	**300**

Suppose the content of PC is 200, while the content of register R1 is 400. XR register is 100 if all the numbers and addresses in decimal numbers determine the content of AC and effective address for the following addressing modes.

(a) Direct Addressing (b) Indirect Addressing (c) Relative Addressing (d) Indexed Addressing(e) Register Indirect Addressing.

Solution :

(1) Direct Addressing: Since it is given that the address field of instruction is 500, so it direct mode, this value itself is an Effective Address.

So Effective Address = 500 and

Operand = 800.

(2) Indirect Mode

Effective Address = M[500]=800

Operand = M[800]= 300

(3) Relative Addressing Mode:

Effective Address = PC + 500 = 202 + 500 =702

Operand =325

(4) Indexed Address Mode

Effective Address = Base Register Value + 500

= 100 + 500 = 600

Operand = M[600] = 900

(5) Register Indirect Mode

Effective Address = M[R1]= 400

Operand = M[400]=700.

11.Register R1 and R2 contain values 1800 and 3800 respectively. The word length of the processor is 4 bytes. What is the effective address of the memory operand in each one of the following cases?

i.ADD 100(R2), R6.
ii.LOAD R6, 20(R1,R2)
iii.STORE –(R2), R6
iv.SUBTRACT (R2)+, R6

Solution:

a) Effective address = 100 + Contents of R2 = 100 + 3800 = 3900.
b) Effective address = 20 + Contents of R1 + Contents of R2 = 20 + 1800 + 3800 = 5620.
c) Effective address = Contents of R2 – 4 = 3800 – 4 = 3796.
d) Effective address = Contents of R2 = 3800.

12.For the following 68000 instructions, determine how many bytes does each instruction occupy? How many memory accesses does the fetching and execution of each instruction require?

i.SUB.L D1, (A1)
ii.SUB.L (A2,D1),D1
iii.SUB.L #$1234,(A1)

Solution:

in the 68000 the OP code of each instruction occupies 16 bits. If the instruction has an immediate operand, and if the operand is 8 or 16 bits, then the instruction occupies one extension word in addition to the OP code word. If the immediate operand is 32 bits, then the instruction occupies 2nd and 3rd extension words. For the indexed and relative addressing modes, the required index value (either as an immediate operand, or in a register) is given in the word that follows the OP code. That is, an instruction that uses indexed and relative addressing modes occupies 2 16-bit words.

The number of memory accesses required to fetch an instruction is equal to the number of words in an instruction. The number of memory accesses to execute an instruction depends on the size of the operand and if the memory operand is used as a source or destination or both.

If the operand size is a long word, and the memory operand is a source operand, two memory accesses are required.

If the operand size is a long word, and the memory operand is a destination operand, two memory accesses are required.

If the operand size is a long word, and the memory operand is a source and destination operand, four memory accesses are required.

a) This instruction occupies 2 bytes (one word), as a result, one memory access is needed to fetch the instruction. It has one memory operand as the source and one memory operand as the destination and the operand size is a long word, so four memory accesses are needed to execute the instruction.
b) This instruction occupies 4 bytes, or 2 16 bit words since it uses the indexed addressing mode. As a result, 2 memory accesses are required to fetch the instruction. It uses one memory operand as a source operand and the operand size is a long word, so two memory access is required.
c) This instruction occupies 6 bytes, or 3 16 bit words since it uses an immediate operand of length 32 bits. 3 memory accesses are needed to fetch the instruction. It uses a memory operand as a source and a destination and the operand size is a long word, so 4 memory accesses are required to execute it.

INSTRUCTION FORMATS

13. A computer has an instruction size of 16 bits and has 16 programmer visible registers. Each instruction has two source and one destination operands and uses only register direct addressing. The maximum number of opcodes that this processor can have is

Concept:

Instruction consists of the following:

- Source – 1
- Source – 2
- Destination
- Opcode [It is not mentioned in question but opcode is obvious for every instruction]

Calculation:

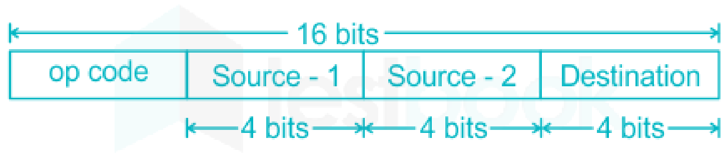
Given:

Number of registers = 16

Instruction size = 16 bits

Number of bits required to represent each register

$n = \log_2 (16) = \log_2 2^4 = 4$ bits.



No. of bits for opcode field = 16 – 12 = 4 bits

∴ The total number opcodes = $2^4 = 16$

INSTRUCTION TYPES:

14. Consider a processor with 64 registers and an instruction set of size twelve. Each instruction has five distinct fields, namely, opcode, two source register identifiers, one destination register identifier, and a twelve-bit immediate value. Each instruction must be stored in memory in a byte-aligned fashion. If a program has 100 instructions, how much amount of memory (in bytes) consumed by the program text?

Explanation: One instruction is divided into five parts,

- **(1)**: The opcode- As we have instruction set of size 12, an instruction opcode can be identified by 4 bits, as 2^4=16 and we cannot go any less.
- **(2)** & **(3)**: Two source register identifiers- As there are total 64 registers, they can be identified by 6 bits. As they are two i.e. 6 bit + 6 bit.
- **(4)**: One destination register identifier- Again it will be 6 bits.
- **(5)**: A twelve bit immediate value- 12 bit.

Adding them all we get,

= 4 + 6 + 6 + 6 + 12

= 34 bit

= 34/8 byte

= 4.25 byte

As given Each instruction must be stored in memory in a byte-aligned fashion,4.25 is not byte alignment, memory address should be 0,1,2,3,4,5,6,7…….. so it should be 5 bytes.

As there are 100 instructions, we have a size of 5*100= 500 bytes.

Hence (D) 500 is the answer.

15. A processor has 16 integer registers (R0, R1, … , R15) and 64 floating point registers (F0, F1, … , F63). It uses a 2 byte instruction format. There are four categories of instructions: Type-1, Type-2, Type-3, and Type 4. Type-1 category consists of four instructions, each with 3 integer register operands (3Rs). Type-2 category consists of eight instructions, each with 2 floating point register operands (2Fs). Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand (1R+1F). Type-4 category consists of N instructions, each with a floating point register operand (1F). What is the maximum value of N ?

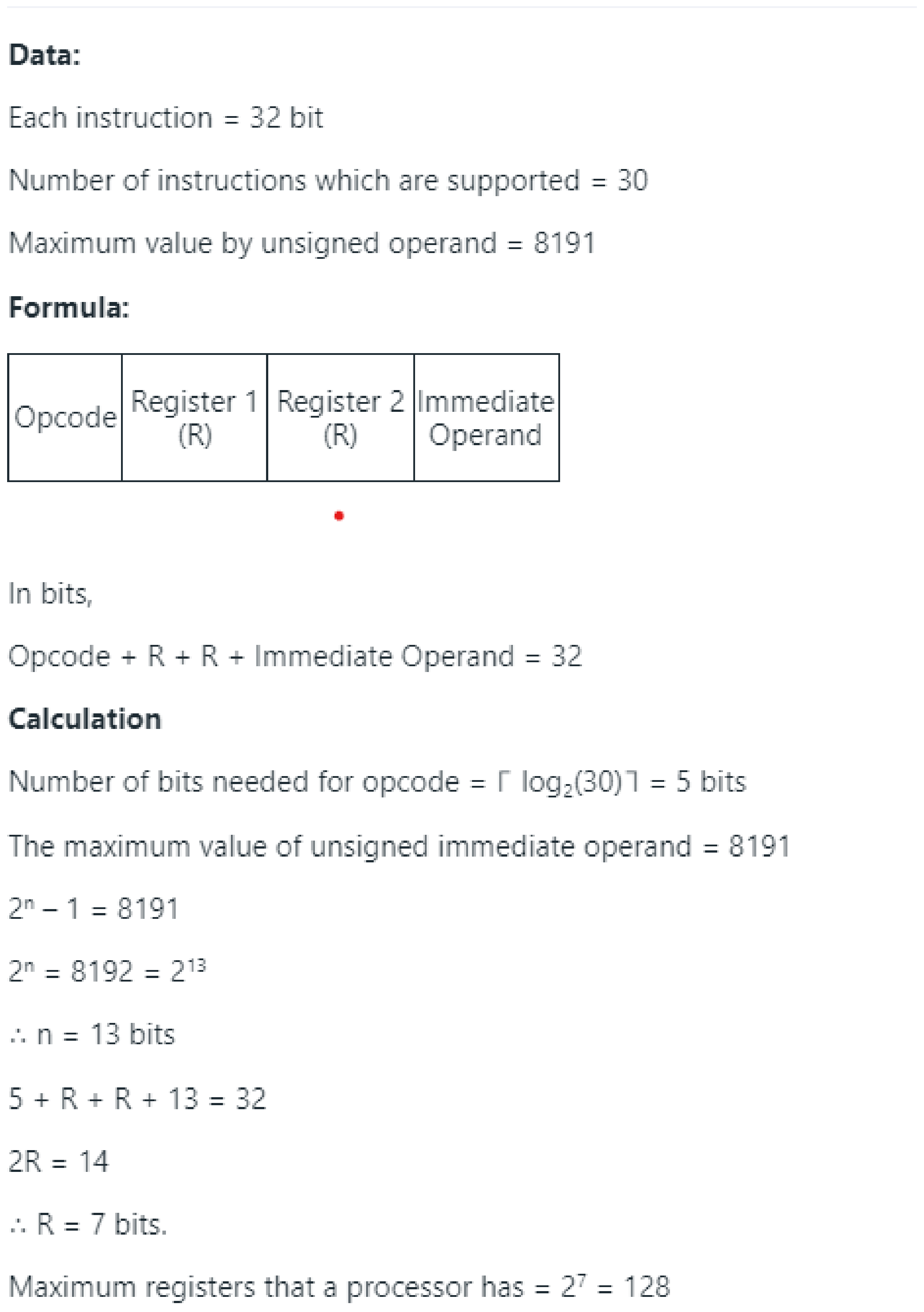
Explanation: Given, size of instruction format is 2 byte (= 16 bits), therefore number of instruction encoding = 2^{16}
Also, total number of bits in integer operand = $\log_2$(16 integer registers) = 4

Total number of bits in floating point operand = $\log_2$(64 floating point registers) = 6
So, number of encoding consumed:

By type 1 instructions = $4\times2^{3\times4} = 2^{14}$
By type 2 instructions = $8\times2^{2\times6} = 2^{15}$
By type 3 instructions = $14\times2^{(4+6)} = 14336$
Now, number of encoding left for type 4 instructions = $2^{16} - (2^{14} + 2^{15} + 14336) = 2048$
Therefore, total number of different instructions of type 4 instructions = 2048 /64 = 32
Please note that there is difference between number of different instructions and number of different encoding, a single instruction can have different encodings when the address part differs.

So, answer is 32.

16. Consider a 32-bit processor which supports 30 instructions. Each instruction is 32 bit long and has 4 fields namely opcode, two register identifiers and an immediate operand of unsigned integer type Maximum value of the immediate operand that can be supported by the processor is 8191. How many registers the processor has?



Data:

Each instruction = 32 bit

Number of instructions which are supported = 30

Maximum value by unsigned operand = 8191

Formula:

Opcode	Register 1 (R)	Register 2 (R)	Immediate Operand

In bits,

Opcode + R + R + Immediate Operand = 32

Calculation

Number of bits needed for opcode = $\lceil \log_2(30) \rceil$ = 5 bits

The maximum value of unsigned immediate operand = 8191

$2^n - 1 = 8191$

$2^n = 8192 = 2^{13}$

$\therefore n = 13$ bits

5 + R + R + 13 = 32

2R = 14

$\therefore$ R = 7 bits.

Maximum registers that a processor has = $2^7 = 128$

17. A CPU has 12 registers and uses 6 addressing modes. RAM is 64K x 32. What is the maximum size of the op- code field if the instruction has a register operand and a memory address operand?

Data:

number of registers = 12

addessing mode = 6

RAM size =64K × 32 = $2^{16} \times 2^{5}$

Formula:

Memory Capacity is of the form = $2^{m} \times 2^{n}$

Address lines required = m

Instrcution size = Data Lines = 2^{n}

number of bits = $\lceil \log_2 n \rceil$

number of register or number addressing modes

Calculation:

Instrcution size = Data Lines = 32

Address lines required = 16 bits

number of bits for a addressing mode = $\lceil \log_2 6 \rceil = 3$

number of bits for a register field = $\lceil \log_2 12 \rceil = 4$

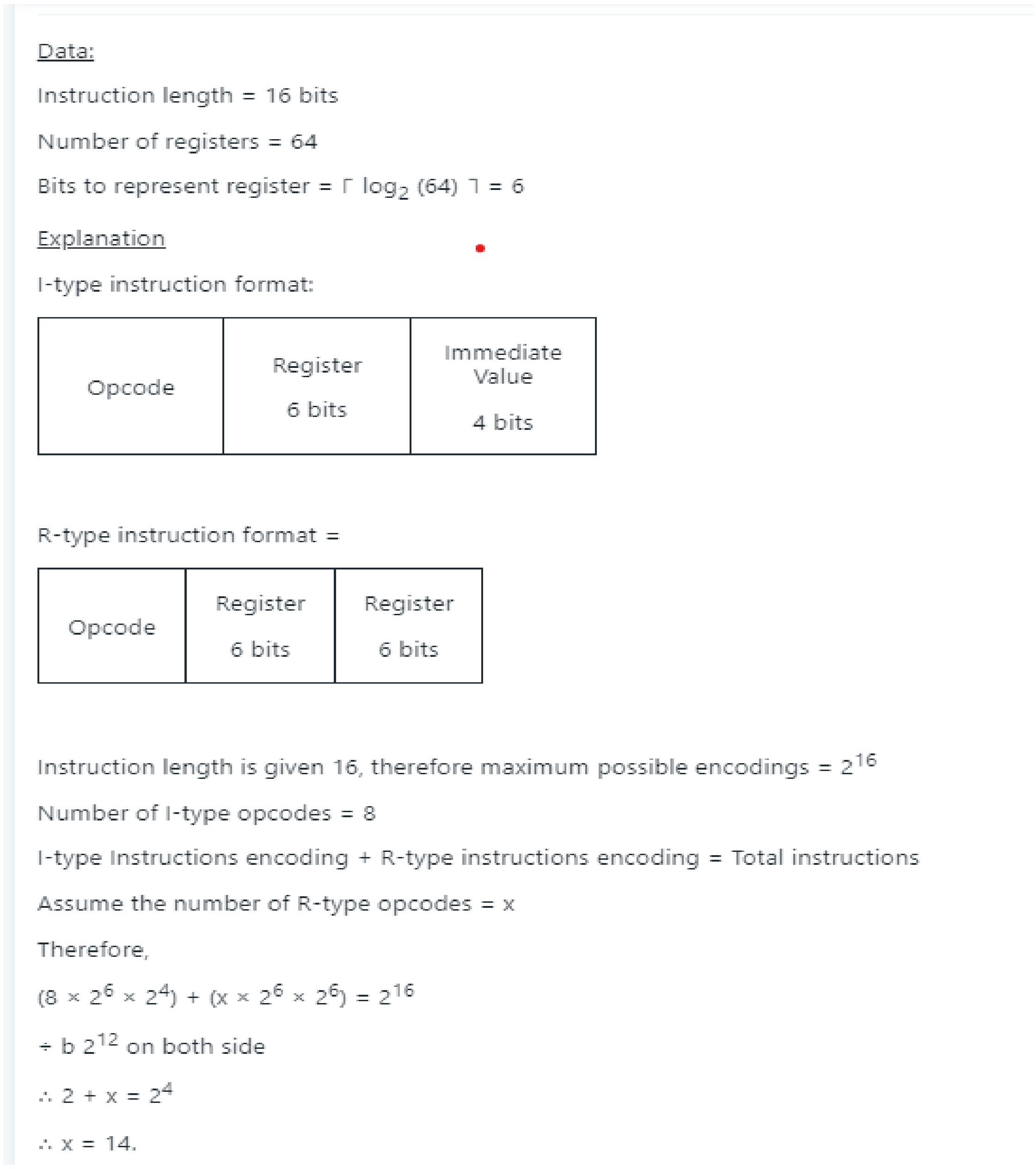
Addressing Mode	op-code field	Register	Memory Address field
3 bits	x bits	4 bits	16 bits

$3 + x + 4 + 16 = 32$

∴ x = 9

op-code field = 9 bits

18. A processor has 64 registers and uses 16-bit instruction format. It has two types of instructions: I-type and R-type. Each I-type instruction contains an opcode, a register name, and a 4-bit immediate value. Each R-type instruction contains an opcode and two register names. If there are 8 distinct I-type opcodes, then the maximum number of distinct R-type opcodes is



Data:

Instruction length = 16 bits

Number of registers = 64

Bits to represent register = $\lceil \log_2 (64) \rceil = 6$

Explanation

I-type instruction format:

Opcode	Register 6 bits	Immediate Value 4 bits

R-type instruction format =

Opcode	Register 6 bits	Register 6 bits

Instruction length is given 16, therefore maximum possible encodings = 2^{16}

Number of I-type opcodes = 8

I-type Instructions encoding + R-type instructions encoding = Total instructions

Assume the number of R-type opcodes = x

Therefore,

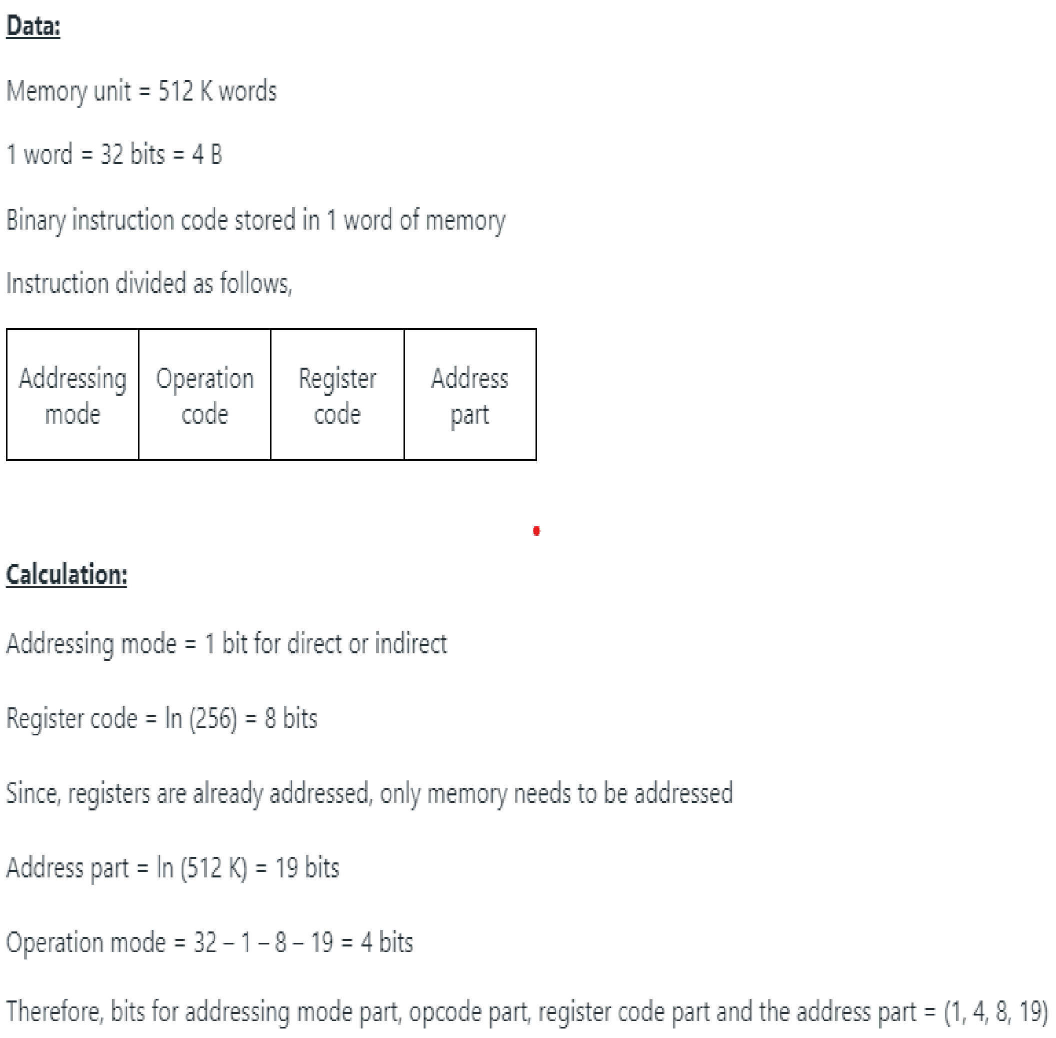
$(8 \times 2^6 \times 2^4) + (x \times 2^6 \times 2^6) = 2^{16}$

÷ b 2^{12} on both side

$\therefore 2 + x = 2^4$

$\therefore x = 14.$

19. A computer uses a memory unit of 512 K words of 32 bits each. A binary instruction code is stored in one word of the memory. The instruction has four parts: an addressing mode field to specify one of the two-addressing mode (direct and indirect), an operation code, a register code part to specify one of the 256 registers and an address part. How many bits are there in addressing mode part, opcode part register code part and the address part?



Data:

Memory unit = 512 K words

1 word = 32 bits = 4 B

Binary instruction code stored in 1 word of memory

Instruction divided as follows,

Addressing mode	Operation code	Register code	Address part

Calculation:

Addressing mode = 1 bit for direct or indirect

Register code = ln (256) = 8 bits

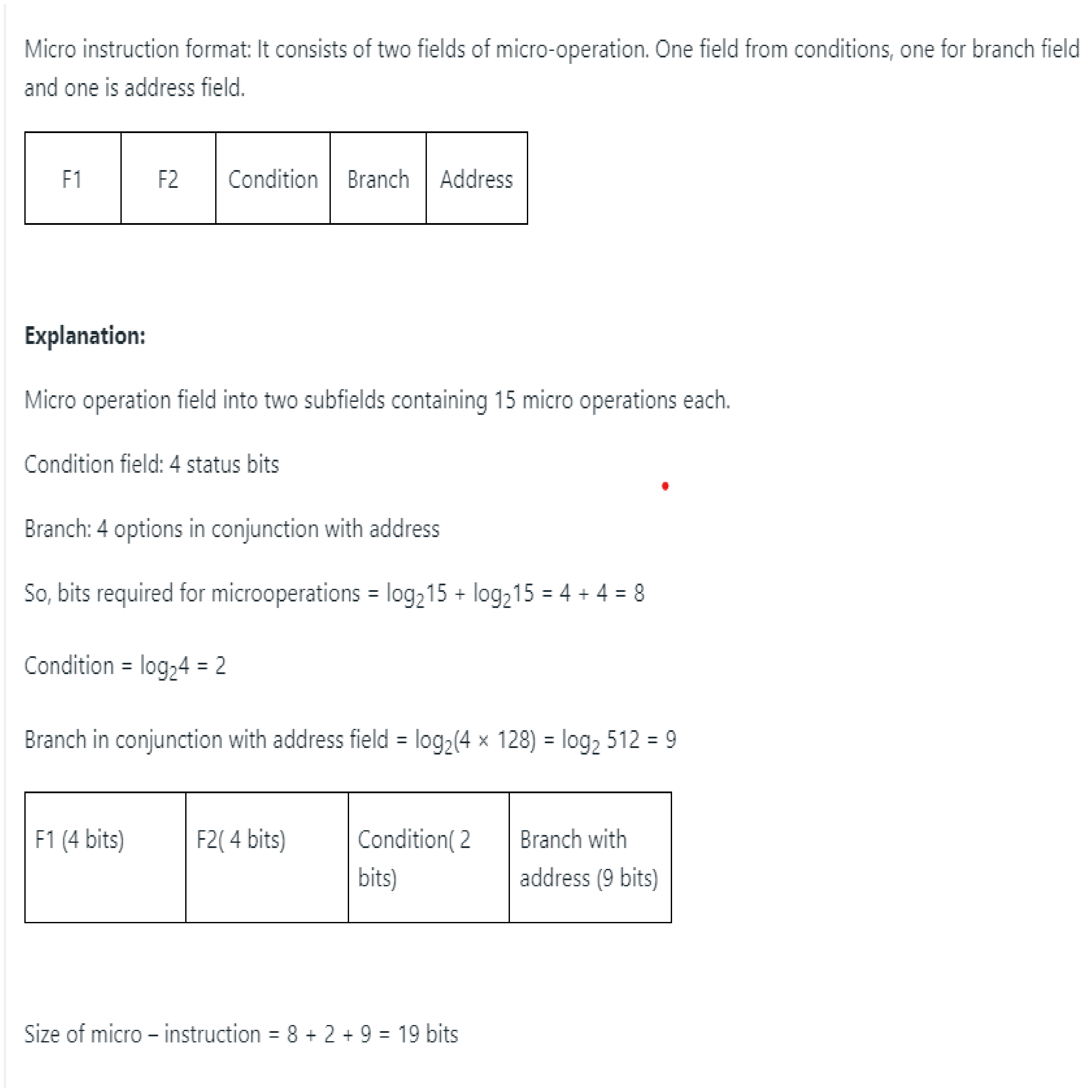
Since, registers are already addressed, only memory needs to be addressed

Address part = ln (512 K) = 19 bits

Operation mode = 32 – 1 – 8 – 19 = 4 bits

Therefore, bits for addressing mode part, opcode part, register code part and the address part = (1, 4, 8, 19)

20. A micro instruction format has micro operation field which is divided into 2 subfields F1 and F2, each having 15 distinct microoperations, condition field CD for four status bits, branch field BR having four options used in conjunction with address field AD. The address space is of 128 memory words. The size of micro instruction is:



Micro instruction format: It consists of two fields of micro-operation. One field from conditions, one for branch field and one is address field.

F1	F2	Condition	Branch	Address

Explanation:

Micro operation field into two subfields containing 15 micro operations each.

Condition field: 4 status bits

Branch: 4 options in conjunction with address

So, bits required for microoperations = $\log_2 15 + \log_2 15 = 4 + 4 = 8$

Condition = $\log_2 4 = 2$

Branch in conjunction with address field = $\log_2(4 \times 128) = \log_2 512 = 9$

F1 (4 bits)	F2(4 bits)	Condition(2 bits)	Branch with address (9 bits)

Size of micro – instruction = 8 + 2 + 9 = 19 bits

21. Consider a 32-bit processor which supports 30 instructions. Each instruction is 32 bit long and has 4 fields namely opcode, two register identifiers and an immediate operand of unsigned integer type Maximum value of the immediate operand that can be supported by the processor is 8191. How many registers the processor has?

The given Data,

Distinct Instructions = 300

General Purpose Registers – 70

Opoce Instruction word – 32 bit

Two register operands and an immediate operand

Explanation:

Each

Instruction has 32 bits. To support 300 instructions, the opcode must contain 9-bits.

Register operand1 requires 7 bits, since the total registers are 70, register operand 2 also requires 7 bits

So, 32-(9+7+7)= 9 bits are left over for immediate operand.

So Answer is 9.

22. A machine has a 32-bit architecture, with 1-word long instructions. It has 64 registers, each of which is 32 bits long. It needs to support 45 instructions, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer, the maximum value of the immediate operand is

Data:

Each instruction = 32 bit

Number of instructions which are supported = 45

Number of registers = 64

Formula:

The maximum value of unsigned immediate operand = $2^n - 1$

where n is the number of bits in immediate operand field

Opcode	Register 1 (R)	Register 2 (R)	Immediate Operand

Opcode + R + R + Immediate Operand = 32 bits

Calculation

Number of bits needed for opcode = $\lceil \log_2(45) \rceil$ = 6 bits

Number of bits needed for registers = $\lceil \log_2(64) \rceil$ = 6 bits

6 + 6 + 6 + Immediate Operand = 32

Immediate Operand = (32 – 18) bits = 14 bits

The maximum value of unsigned immediate operand = $2^{14} - 1 = 16383$

DAY 3 :

Problems on Integer Arithmetic - Addition and Subtractions

1. Add **1101 and -1001**

    - First, find the 2's complement of the negative number 1001. So, for finding 2's complement, change all 0 to 1 and all 1 to 0 or find the 1's complement of the number 1001. The 1's complement of the number 1001 is 0110, and add 1 to the LSB of the result 0110. So the 2's complement of number 1001 is 0110+1=0111
    - Add both the numbers, i.e., 1101 and 0111; 1101+0111=1 0100
    - By adding both numbers, we get the end-around carry 1. We discard the end-around carry. So, the addition of both numbers is 0100.

2. 10101 – 00111

We take 2's complement of subtrahend 00111, which is 11001. Now, sum them. So,

10101+11001 =1 01110.

In the above result, we get the carry bit 1. So we discard this carry bit and remaining is the final result and a positive number.

3. 10101 - 10111

- take 2's complement of subtrahend 10111, which comes out 01001. Now, we add both of the numbers. So,
- 10101+01001 =11110.
- In the above result, we didn't get the carry bit. So calculate the 2's complement of the result, i.e., 00010. It is the negative number and the final answer.

4. Add x=2, y=3

```
addition: 0010
                 0011
            -----------
                0101
             ----------         +5 in 4 bits
```

5. x=-2,y=-3

```
addition: 1110
          1101
          1011
---> -5 in 4 bits
```

6. x=3, y=2 for 4 bits 2's complement representation

3---> 0011

Take 2's complement of 2----> 0010

1101 + 1 ---> 1110

Adding 0011 and 1110

0011

1110

0001

---> +1 (discard carry 1 from MSB)

7. add $(8)_{10}$ and $(-3)_{10}$.

First we have to convert them into 2's complement and simply add them.

$$(8)_{10} \Rightarrow (00001000)_2$$
$$(-3)_{10} \Rightarrow (11111101)_2$$
$$Now, (8)_{10} + (-3)_{10} = (+5)_{10}$$

```
Now 0000 1000
    1111 1101
-----------------------
Carry 1 ← 0000 0101 = (+5)10
```

8. subtract $(8)_{10}$ from $(9)_{10}$.

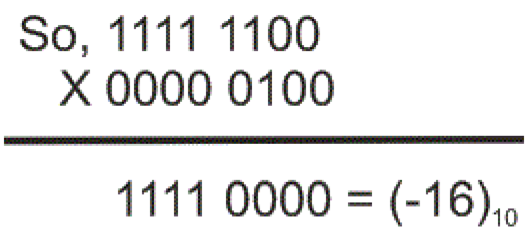
First convert $(8)_{10}$ from $(-9)_{10}$ into 2's complement and simply add.

$$(8)_{10} \Rightarrow (00001000)2$$
$$(-9)_{10} \Rightarrow (11110111)2$$
$$So,\ (8)_{10} - (9)_{10} = (8)_{10} + (-9)_{10} = (-1)_{10}$$

```
Now 0000 1000
    1111 0111
-----------------------
Carry 0 ← 1111 1111 = (-1)10
```

9. multiply $(-4)_{10}$ with $(4)_{10}$

Now , $(-4)_{10}$ = 1111 1100 in 2's complement and $(4)_{10}$ = 0000 0100 in 2's complement



So, 1111 1100
X 0000 0100

1111 0000 = $(-16)_{10}$

10. Using addition with 2s complement, calculate the sum of 1101 and -1110.

Ans: -0001

11. Add 0011 and -0101.

Ans: -0010

12. Find the addition with 2s complement of a negative number, -8 and -5.Ans: -1101

In a 5-bit register find the sum of the following by using 2's complement:

-1011 and -0101

Solution:

+ 1 0 1 1 ⇒ $\underline{0}$ 1 0 1 1

- 0 1 0 1 ⇒ $\underline{1}$ 1 0 1 1 (2's complement)

(Carry 1 discarded) $\underline{0}$ 0 1 1 0

Hence the sum is + 0110.

13. **In a 5-bit register find the sum of the following by using 2's complement:**

+ 0111 and – 0011.

Solution:

$$\begin{array}{lcl} +0111 & \Rightarrow & \underline{0}\,0111 \\ -0011 & \Rightarrow & \underline{1}\,1101 \\ \text{(Carry 1 discarded)} & & \underline{0}\,0100 \end{array}$$

Hence the sum is + 0100.

14. In a 5-bit register find the sum of the following by using 2's complement:

(i) + 0 0 1 1 and - 0 1 0 1

Solution:

$$\begin{array}{lcll} +0011 & \Rightarrow & \underline{0}\,0011 & \\ -0101 & \Rightarrow & \underline{1}\,1011 & \text{(2's complement)} \\ & & \underline{1}\,1110 & \end{array}$$

2's complement of 1110 is (0001 + 0001) or 0010.

Hence the required sum is - 0010.

15. + 0 1 0 0 and - 0 1 1 1

Solution:

+ 0 1 0 0 ⇒ $\underline{0}$ 0 1 0 0

- 0 1 1 1 ⇒ $\underline{1}$ 1 0 0 1 (2’s complement)

$\underline{1}$ 1 1 0 1

2’s complement of 1101 is 0011.

Hence the required sum is – 0011.

16. In a 5-bit register find the sum of the following by using 2’s complement:

– 0011 and – 0101

Solution:

- 0 0 1 1 ⇒ $\underline{1}$ 1 1 0 1 (2’s complement)

- 0 1 0 1 ⇒ $\underline{1}$ 1 0 1 1 (2’s complement)

(Carry 1 discarded) $\underline{1}$ 1 0 0 0

2’s complement of 1000 is (0111 + 0001) or 1000.

Hence the required sum is – 1000.

17. sum **-0111 and – 0010.**

Solution:

- 0 1 1 1 $\Rightarrow$ $\underline{1}$ 1 0 0 1 (2’s complement)

- 0 0 1 0 $\Rightarrow$ $\underline{1}$ 1 1 1 0 (2’s complement)

(Carry 1 discarded) $\underline{1}$ 0 1 1 1

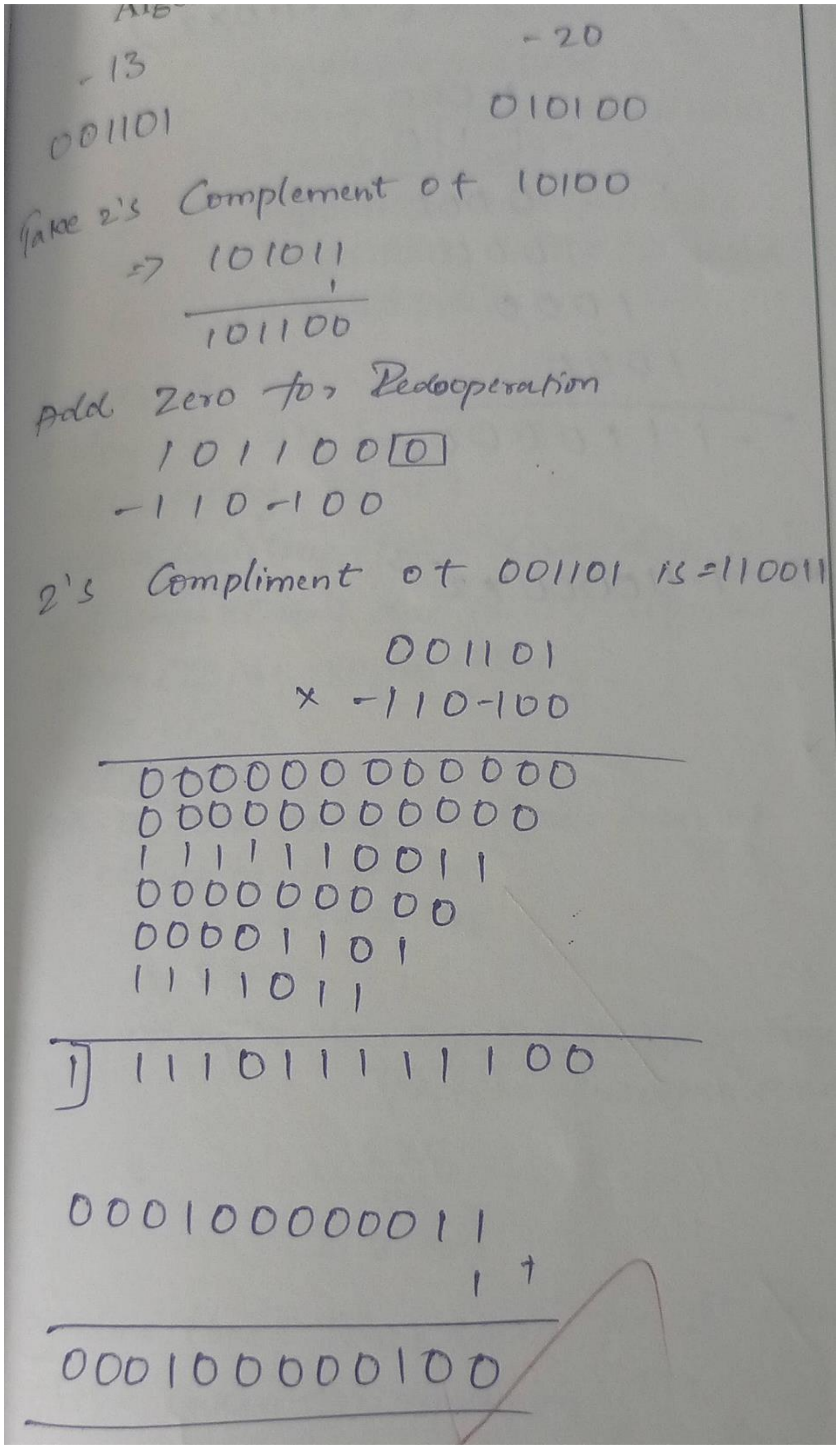
2’s complement of 0111 is 1001.

Hence the required sum is – 1001.

DAY 4

<u>BOOTH ALGORITHM:</u>

1. Perform the multiplication of (-13) and (-20) using Booth Algorithm.



```
-13                          -20
001101                       010100

Take 2's Complement of 10100
  => 101011
          1
     ------
     101100

Add Zero for Redooperation
   1 0 1 1 0 0 [0]
  -1 1 0 -1 0 0

2's Compliment of 001101 is =110011

          0 0 1 1 0 1
        × -1 1 0 -1 0 0
   ---------------------------
    0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0
    1 1 1 1 1 1 0 0 1 1
    0 0 0 0 0 0 0 0 0
    0 0 0 0 1 1 0 1
    1 1 1 1 0 1 1
   ---------------------------
 [1] 1 1 1 0 1 1 1 1 1 1 0 0

  0 0 0 1 0 0 0 0 0 0 1 1
                      1 +
  -----------------------
  0 0 0 1 0 0 0 0 0 1 0 0
```

2. Multiply -5 and -7 by Binary Multiplication method and Recoding principle of Booth Algorithm.

BR = -5 = 1011,
BR' = 0100, <-- 1's Complement (change the values 0 to 1 and 1 to 0)
BR'+1 = 0101 <-- 2's Complement (add 1 to the Binary value obtained after 1's complement)
QR = -7 = 1001 <-- 2's Complement of 0111 (7 = 0111 in Binary)
The explanation of first step is as follows: Qn+1
AC = 0000, QR = 1001, Qn+1 = 0, SC = 4
Qn Qn+1 = 10
So, we do AC + (BR)'+1, which gives AC = 0101
On right shifting AC and QR, we get
AC = 0010, QR = 1100 and Qn+1 = 1

OPERATION	**AC**	**QR**	**Qn+1**	**SC**
	0000	**1001**	**0**	**4**
AC + BR’ + 1	**0101**	**1001**	**0**	
ASHR	**0010**	**1100**	**1**	**3**
AC + BR	**1101**	**1100**	**1**	
ASHR	**1110**	**1110**	**0**	**2**
ASHR	**1111**	**0111**	**0**	**1**
AC + BR’ + 1	**0100**	**0111**	**0**	
ASHR	**0010**	**0011**	**1**	**0**

Product is calculated as follows:
Product = AC QR
Product = 0010 0011 = 35

3. **Multiply the two numbers 23 and -9 by using the Booth's multiplication algorithm.**

Here, M = 23 = (010111) and Q = -9 = (110111)

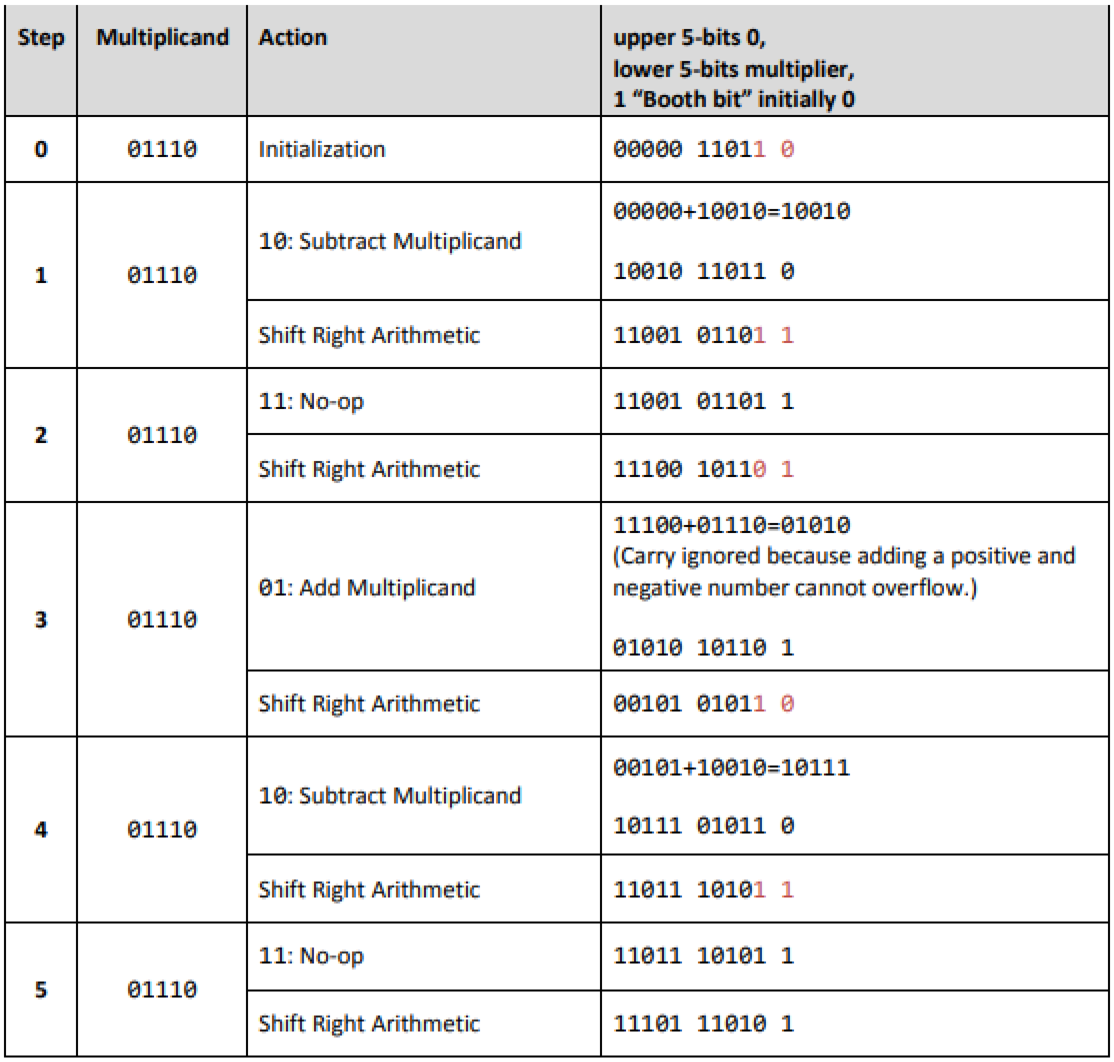
Q_n Q_{n+1}	**M = 0 1 0 1 1 1 M' + 1 = 1 0 1 0 0 1**	**AC**	**Q**	**Q_{n+1} SC**
	Initially	000000	110111	0 6
1 0	Subtract M	101001		
		101001		
	Perform Arithmetic right shift operation	110100	111011	1 5
1 1	Perform Arithmetic right shift operation	111010	011101	1 4
1 1	Perform Arithmetic right shift operation	111101	001110	1 3
0 1	Addition (A + M)	010111		
		010100		
	Perform Arithmetic right shift	001010	000111	0 2

	operation			
1 0	Subtract M	101001		
		110011		
	Perform Arithmetic right shift operation	111001	100011	1 1
1 1	Perform Arithmetic right shift operation	**111100**	**110001**	**1** 0

Q_{n+1} = 1, it means the output is negative.

Hence, 23 * -9 = 2's complement of 111100110001 => **(00001100111)**

4. Multiply 14 * -5



Step	Multiplicand	Action	upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010 10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1
3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1
		Shift Right Arithmetic	00101 01011 0
4	01110	10: Subtract Multiplicand	00101+10010=10111 10111 01011 0
		Shift Right Arithmetic	11011 10101 1
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11101 11010 1

5. Multiply 3 * -25 using 6-bit numbers

Answer:

3_{10} = 00 0011_2

-25_{10} = 10 0111_2

Binary Multiplicand: 000011 Binary Multiplier: 100111 Binary Word Length (n-bit):

Multiplier		Booth Multiplier
Bit[i]	Bit[i-1]	
0	0	0
0	1	+1
1	0	-1
1	1	0

Booth Multiplier Recoding Table:

```
        Multiplier: 1   0   0   1   1   1   0
Booth Recoding:     -1  0  +1   0   0  -1
```

*A red zero is added after the least significant bit (LSB) for the conversion
Click on the zeros in "Booth Recoding" above to view the pair of bit of each conversion!

```
          00  001 1
      ×  -10+100-1
   ------------------
 111111 11  110 1
 000000 01  1
 ------------------
 000000 01  010 1
 111110 1
 ------------------
 111110 11  010 1
```

Booth Multiplier		Bit-Pair Recoding Multiplier	
Bit[i]	Bit[i-1]	Bit[i]	Bit[i-1]
1	-1	0	1
-1	1	0	-1

1	0	0	2
-1	0	0	-2

Bit-Pair Recoding Table:

```
          Multiplier: 1  0  0   1  1  1  0
     Booth Recoding: | -1  0 | +1  0 |  0 -1 |
  Bit-Pair Recoding: |    -2 |    +2 |    -1 |
```

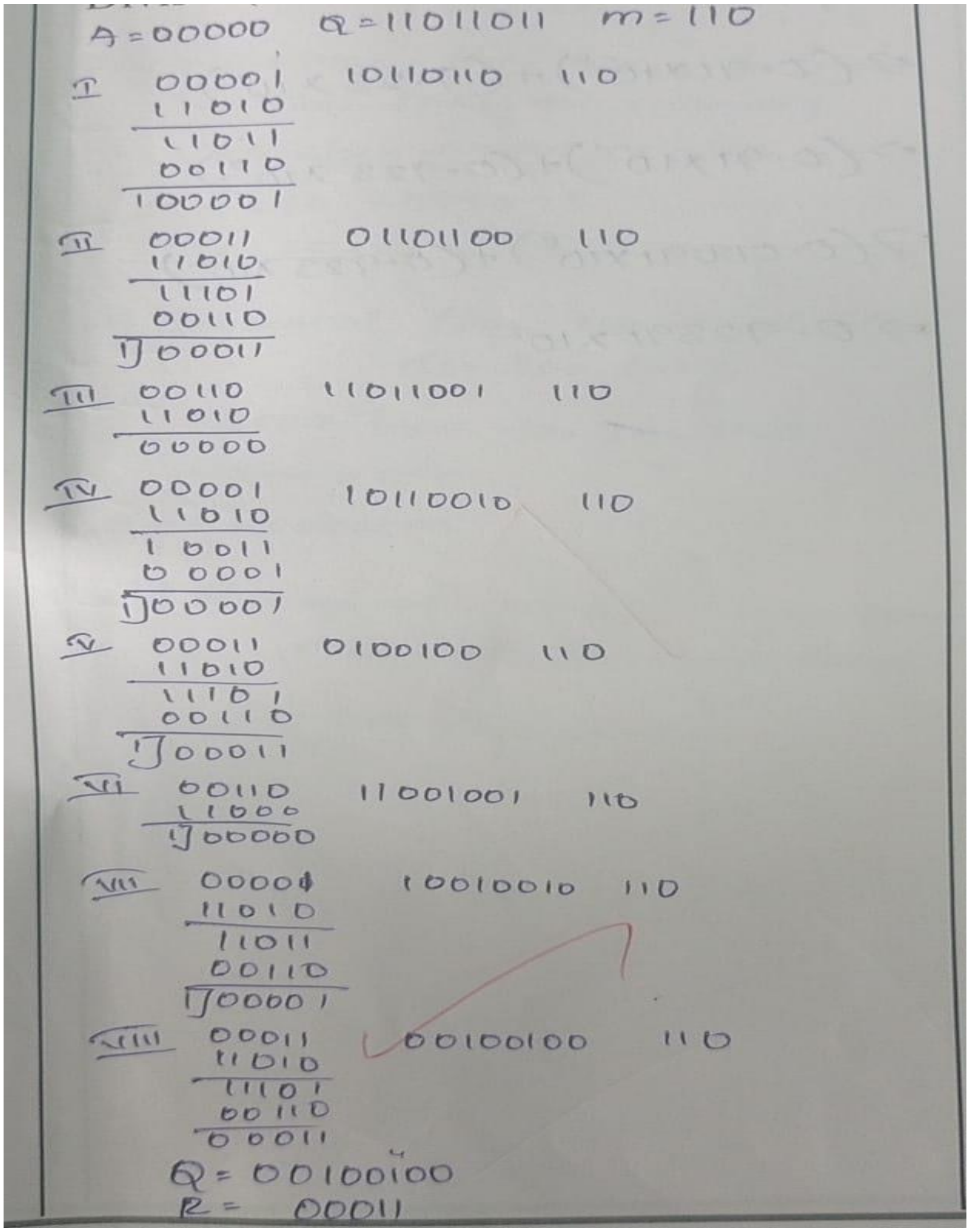
If the Multiplier is an odd number of bits, a 1/0 bit is added to extent the multiplier to an even number of bits before the most significant bit (MSB) for the Bit-Pair Recoding Method conversion. Since the Multiplier is an even number of bits, we don't add the bit before MSB.
Same as the Booth Recoding above, a red zero is added after the least significant bit (LSB) for the Booth Recoding conversion

```
          0 00  01 1
        × 0-20+20-1
        -----------
11111111 11  10 1
00000000 11  0
-----------------
00000000 10  10 1
11111101 0
-----------------
11111101 10  10 1
```

7. Perform the multiplication of (-7) and (+3) by Binary Multiplication method and Recoding principle of Booth Algorithm.

Integer Restoring & Non Restoring Division

8. Perform Integer Division using Restoration Method for the Dividend (Q) = 11011011 and the Divisor (M)= 110.



```
A = 00000       Q = 11011011      m = 110

I     00001      10110110     110
      11010
     ------
      11011
      00110
     ------
     100001

II    00011      01101100     110
      11010
     ------
      11101
      00110
     ------
     1]00011

III   00110      11011001     110
      11010
     ------
      00000

IV    00001      10110010     110
      11010
     ------
     1 0011
     0 0001
     ------
     1]00001

V     00011      0100100      110
      11010
     ------
      11101
      00110
     ------
     1]00011

VI    00110      11001001     110
      11000
     ------
     1]00000

VII   00001      10010010     110
      11010
     ------
      11011
      00110
     ------
     1]00001

VIII  00011      00100100     110
      11010
     ------
      11101
      00110
     ------
      00011

Q = 00100100
R =   00011
```

9. Perform Division Restoring Algorithm
Dividend = 11
Divisor = 3

n	**M**	**A**	**Q**	**Operation**
4	00011	00000	1011	initialize
	00011	00001	011_	shift left AQ
	00011	11110	011_	A=A-M
	00011	00001	0110	Q[0]=0 And restore A
3	00011	00010	110_	shift left AQ
	00011	11111	110_	A=A-M
	00011	00010	1100	Q[0]=0
2	00011	00101	100_	shift left AQ
	00011	00010	100_	A=A-M
	00011	00010	1001	Q[0]=1
1	00011	00101	001_	shift left AQ
	00011	00010	001_	A=A-M
	00011	00010	0011	Q[0]=1

Remember to restore the value of A most significant bit of A is 1. As that register Q contain the quotient, i.e. 3 and register A contain remainder 2.

10.Perform the Division using Restoring and Non Restoring Division for the Dividend=1010 and Divisor = 0011

11.Perform Non_Restoring Division for Unsigned Integer

Dividend =11

Divisor =3

-M =11101

N	M	A	Q	Action
4	00011	00000	1011	Start
		00001	011_	Left shift AQ
		11110	011_	A=A-M
3		11110	0110	Q[0]=0
		11100	110_	Left shift AQ
		11111	110_	A=A+M

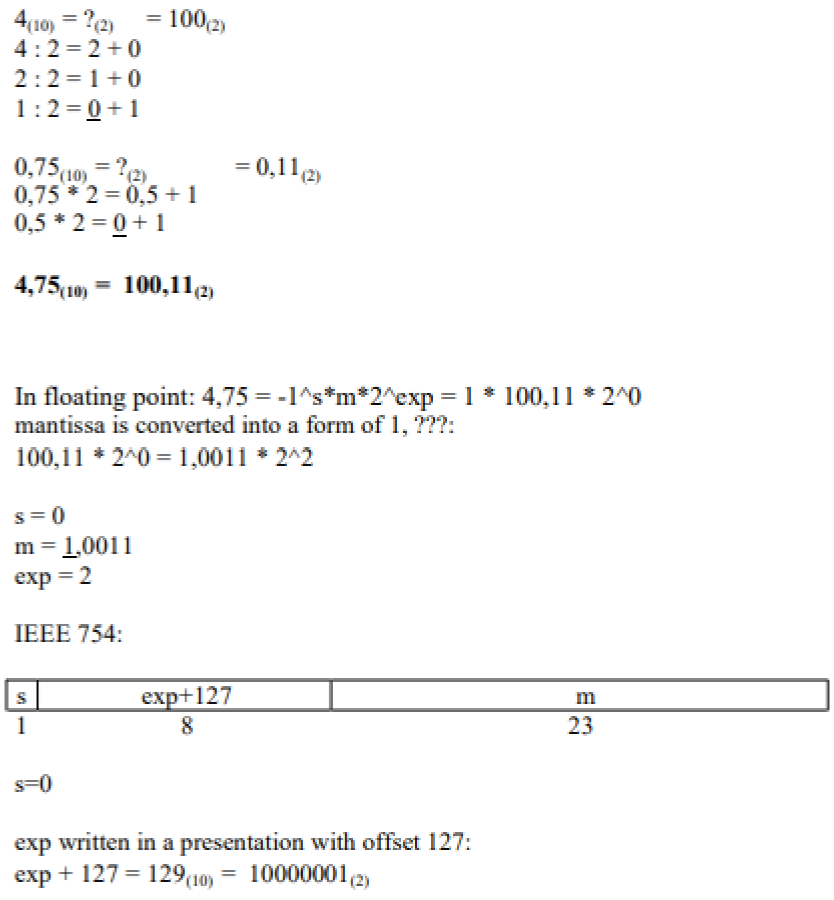
N	M	A	Q	Action
2		11111	1100	Q[0]=0
		11111	100_	Left Shift AQ
		00010	100_	A=A+M
1		00010	1001	Q[0]=1
		00101	001_	Left Shift AQ
		00010	001_	A=A-M
0		00010	0011	Q[0]=1

Quotient = 3 (Q)

Remainder = 2 (A)

FLOATING POINT OPERATIONS:

12. Add the following two decimal numbers in scientific notation (0.91×10^{-1}) with (9.23×10^{1})
13. Multiply the following two numbers in scientific notation. (1.110×10^{10}) with (9.200×10^{-5})
14. Multiply (1.000×2^{-1}) with (-1.110×2^{-2})
15. Write the value of 4.75 in 32-bit floating point format IEEE 754. Another floating point value is written in the memory as 44FAC000(16). Which value does it represent if it is written in the format IEEE 754? (This problem is informative, similar floating point problems will not appear in written exams)



$4_{(10)} = ?_{(2)} \quad = 100_{(2)}$
$4 : 2 = 2 + 0$
$2 : 2 = 1 + 0$
$1 : 2 = \underline{0} + 1$

$0{,}75_{(10)} = ?_{(2)} \quad = 0{,}11_{(2)}$
$0{,}75 * 2 = \underline{0}{,}5 + 1$
$0{,}5 * 2 = \underline{0} + 1$

$\mathbf{4{,}75_{(10)} = 100{,}11_{(2)}}$

In floating point: 4,75 = -1^s*m*2^exp = 1 * 100,11 * 2^0
mantissa is converted into a form of 1, ???:
100,11 * 2^0 = 1,0011 * 2^2

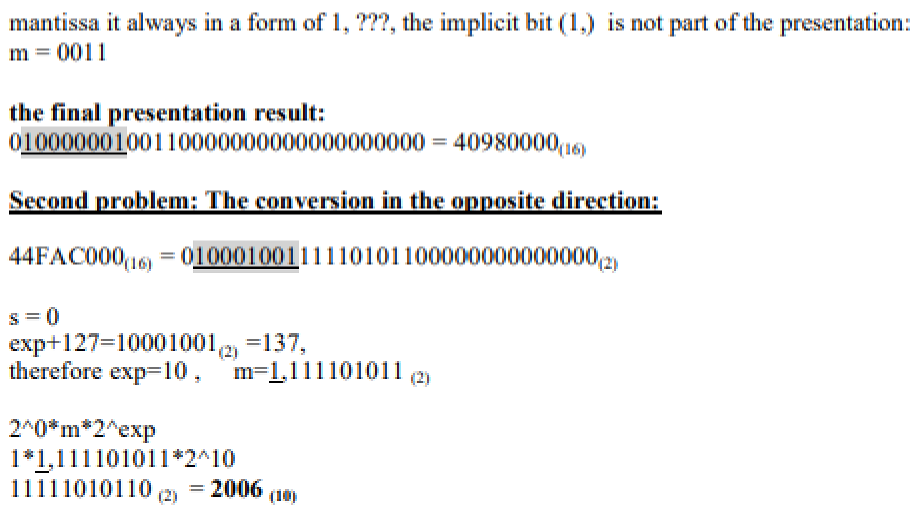
s = 0
m = $\underline{1}$,0011
exp = 2

IEEE 754:

s	exp+127	m
1	8	23

s=0

exp written in a presentation with offset 127:
exp + 127 = $129_{(10)} = 10000001_{(2)}$


mantissa it always in a form of 1, ???, the implicit bit (1,) is not part of the presentation:
m = 0011

the final presentation result:
01000000100110000000000000000000 = 40980000$_{(16)}$

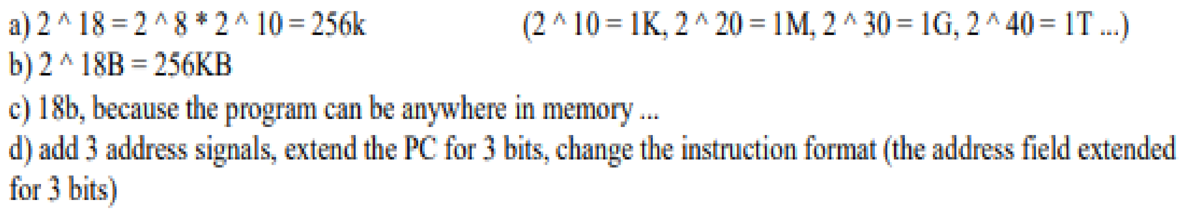
Second problem: The conversion in the opposite direction:

44FAC000$_{(16)}$ = 01000100111111010110000000000000$_{(2)}$

s = 0
exp+127=10001001$_{(2)}$ =137,
therefore exp=10 ,   m=1,111101011 $_{(2)}$

2^0*m*2^exp
1*1,111101011*2^10
11111010110 $_{(2)}$ = **2006** $_{(10)}$

16. Minicomputers in the eighties (eg. DEC PDP-11) had 18 address signals and of course, the 18-bit address bus. Answer the following questions: a) What was the address space of these computers? b) What may have been the largest possible memory of these computers in bytes if the memory location is 1 Byte? c) How long should the program counter (PC) of these computers be? d) What would be needed to change in these computers if we would like to increase address space 8-times?\


a) 2 ^ 18 = 2 ^ 8 * 2 ^ 10 = 256k   (2 ^ 10 = 1K, 2 ^ 20 = 1M, 2 ^ 30 = 1G, 2 ^ 40 = 1T ...)
b) 2 ^ 18B = 256KB
c) 18b, because the program can be anywhere in memory ...
d) add 3 address signals, extend the PC for 3 bits, change the instruction format (the address field extended for 3 bits)

17. Write down a signed decimal number with a value of -25 in 8-bit fixed point presentation in each of the four 8-bit modes for the presentation of signed numbers. Do the same also with the number 33. Write resulting presentations as binary and hexadecimal numbers. Consider, how to perform addition of these numbers in binary form for each presented mode.

a) sign and magnitude:

$V(b) = (-1)^{b_{n-1}} \sum_{i=0}^{n-2} b_i 2^i$ + .. 0 / - .. 1 (-127 .. 127)

$25_{(10)} = ?_{(2)}$ $= 11001_{(2)}$
25 : 2 = 12 + 1
12 : 2 = 6 + 0
6 : 2 = 3 + 0
3 : 2 = 1 + 1
1: 2 = $\underline{0}$ + 1

$33_{(10)} = ?_{(2)}$ $= 100001_{(2)}$
33 : 2 = 16 + 1
16 : 2 = 8 + 0
8 : 2 = 4 + 0
4 : 2 = 2 + 0
2 : 2 = 1 + 0
1 : 2 = $\underline{0}$ + 1

therefore: $-25_{(10)} = 10011001_{(2)}$ ($99_{(16)}$) $33_{(10)} = 00100001_{(2)}$ ($21_{(16)}$)

addition: it is necessary to take into account the sign => additional complexity...

b) the presentation with offset

$V(b) = \sum_{i=0}^{n-1} b_i 2^i - 2^{n-1}$ tudi $(2^{n-1} - 1)$ -128 .. 127 (-127..128)

$-25+2^{n-1} = -25 + 128 = 103$

$103_{(10)} = ?_{(2)}$ $= 1100111_{(2)}$
103 : 2 = 51 + 1
51 : 2 = 25 + 1
25 : 2 = 12 + 1
12 : 2 = 6 + 0
6 : 2 = 3 + 0
3 : 2 = 1 + 1
1 : 2 = $\underline{0}$ + 1

$3+2^{n-1} = 33 + 128 = 161$

therefore: $-25_{(10)} = 01100111_{(2)}$ ($67_{(16)}$) $33_{(10)} = 10100001_{(2)}$ ($A1_{(16)}$)

addition: the result includes two offsets, so one offset has to be subtracted separately \

c) ones' complement

$V(b) = \sum_{i=0}^{n-1} b_i 2^i - b_{n-1}(2^n - 1)$ -127 .. 127

$25_{(10)} = 11001_{(2)}$

therefore: $-25_{(10)} = 11100110_{(2)}$ ($E6_{(16)}$), $33_{(10)} = 00100001_{(2)}$ ($21_{(16)}$)

addition: in case of carry from place *n-1* , 1 has to be added to result; on the other hand, sign bit can be treated the same as the other (value) bits ...

d) two's complement

$V(b) = \sum_{i=0}^{n-1} b_i 2^i - b_{n-1}(2^n)$ -128 .. 127

$25_{(10)} = 00011001_{(2)}$

$11100110_{(2)}$
$+00000001_{(2)}$
$=11100111_{(2)}$

therefore: $-25_{(10)} = 11100111_{(2)}$ ($E7_{(16)}$) $33_{(10)} = 00100001_{(2)}$

PIPELINING STAGES:

18. Consider a pipeline having 4 phases with duration 30, 40, 80 and 70 ns. Given latch delay is 15 ns. Calculate-

1. Pipeline cycle time
2. Non-pipeline execution time
3. Speed up ratio
4. Pipeline time for 500 tasks
5. Sequential time for 500 tasks
6. Throughput

Given-

- Four stage pipeline is used
- Delay of stages = 60, 50, 90 and 80 ns
- Latch delay or delay due to each register = 10 ns

Part-01: Pipeline Cycle Time-

Cycle time

= Maximum delay due to any stage + Delay due to its register

= Max { 60, 50, 90, 80 } + 10 ns

= 90 ns + 10 ns

= 100 ns

Part-02: Non-Pipeline Execution Time-

Non-pipeline execution time for one instruction

= 60 ns + 50 ns + 90 ns + 80 ns

= 280 ns

Part-03: Speed Up Ratio-

Speed up

= Non-pipeline execution time / Pipeline execution time

= 280 ns / Cycle time

= 280 ns / 100 ns

= 2.8

Part-04: Pipeline Time For 1000 Tasks-

Pipeline time for 1000 tasks

= Time taken for 1st task + Time taken for remaining 999 tasks

= 1 x 4 clock cycles + 999 x 1 clock cycle

= 4 x cycle time + 999 x cycle time

= 4 x 100 ns + 999 x 100 ns

= 400 ns + 99900 ns

= 100300 ns

Part-05: Sequential Time For 1000 Tasks-

Non-pipeline time for 1000 tasks

= 1000 x Time taken for one task

= 1000 x 280 ns

= 280000 ns

Part-06: Throughput-

Throughput for pipelined execution

= Number of instructions executed per unit time

= 1000 tasks / 100300 ns

16. Consider a pipeline having 4 phases with duration 50, 60, 75 and 65 ns. Given latch delay is 20 ns. Calculate-

1. Pipeline cycle time
2. Non-pipeline execution time
3. Speed up ratio
4. Pipeline time for 800 tasks
5. Sequential time for 800 tasks
6. Throughput

17. A four stage pipeline has the stage delays as 150, 120, 160 and 140 ns respectively. Registers are used between the stages and have a delay of 5 ns each. Assuming constant clocking rate, the total time taken to process 1000 data items on the pipeline will be- 1. 120.4 microseconds 2. 160.5 microseconds 3. 165.5 microseconds 4. 590.0 microseconds

Solution-

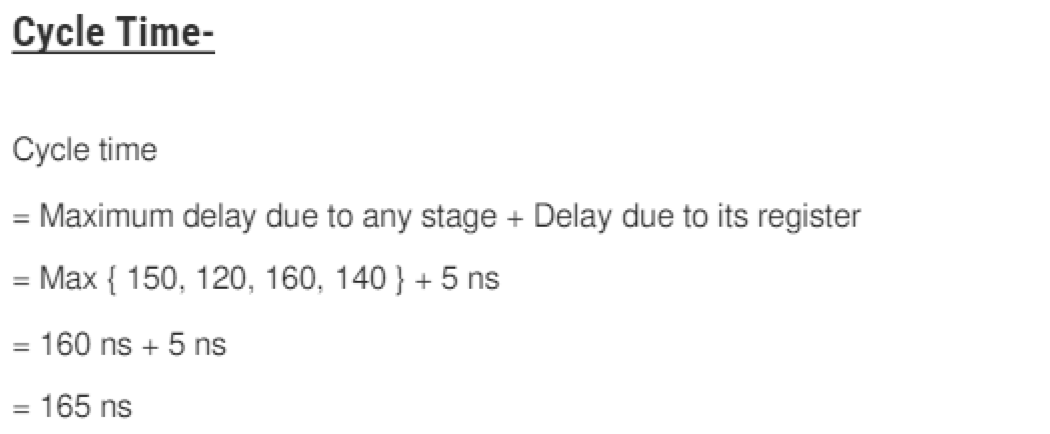
Given-

• Four stage pipeline is used

• Delay of stages = 150, 120, 160 and 140 ns

• Delay due to each register = 5 ns

• 1000 data items or instructions are processed



Cycle Time-

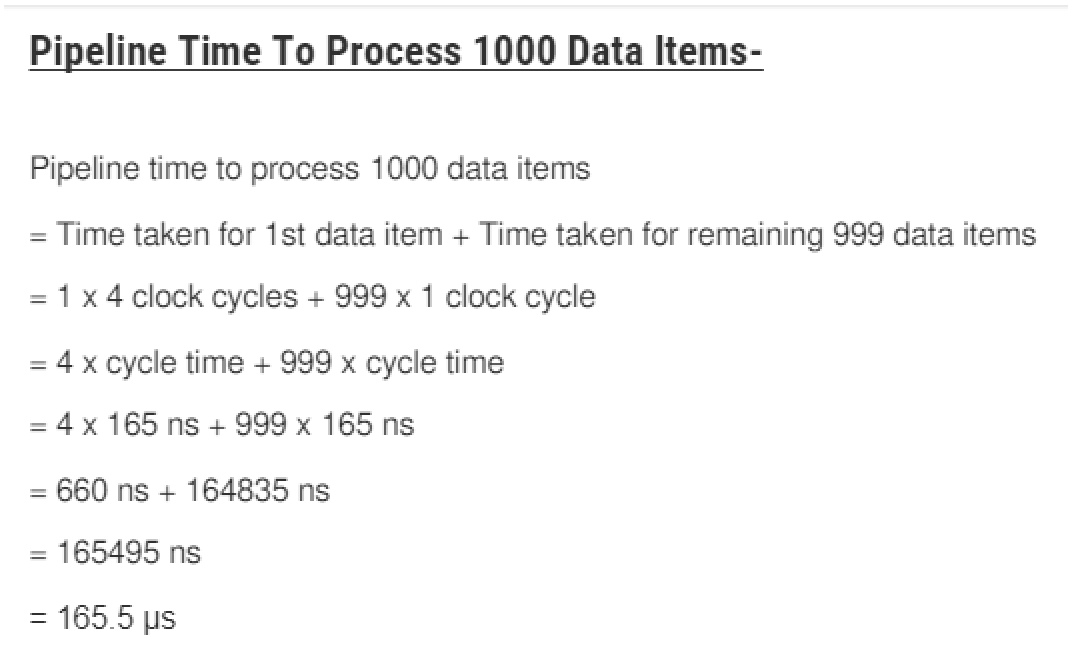
Cycle time

= Maximum delay due to any stage + Delay due to its register

= Max { 150, 120, 160, 140 } + 5 ns

= 160 ns + 5 ns

= 165 ns



Pipeline Time To Process 1000 Data Items-

Pipeline time to process 1000 data items

= Time taken for 1st data item + Time taken for remaining 999 data items

= 1 x 4 clock cycles + 999 x 1 clock cycle

= 4 x cycle time + 999 x cycle time

= 4 x 165 ns + 999 x 165 ns

= 660 ns + 164835 ns

= 165495 ns

= 165.5 μs

18. Consider a non-pipelined processor with a clock rate of 2.5 gigahertz and average cycles per instruction of 4. The same processor is upgraded to a pipelined processor with five stages but due to the internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume there are no stalls in the pipeline. The speed up achieved in this pipelined processor is- 3.2, 3.0, 2.2, 2.0

Cycle Time in Non-Pipelined Processor-

Frequency of the clock = 2.5 gigahertz

Cycle time

= 1 / frequency

= 1 / (2.5 gigahertz)

= 1 / (2.5 x 10^9 hertz)

= 0.4 ns

Non-Pipeline Execution Time-

Non-pipeline execution time to process 1 instruction

= Number of clock cycles taken to execute one instruction

= 4 clock cycles

= 4 x 0.4 ns

= 1.6 ns

Cycle Time in Pipelined Processor-

Frequency of the clock = 2 gigahertz

Cycle time

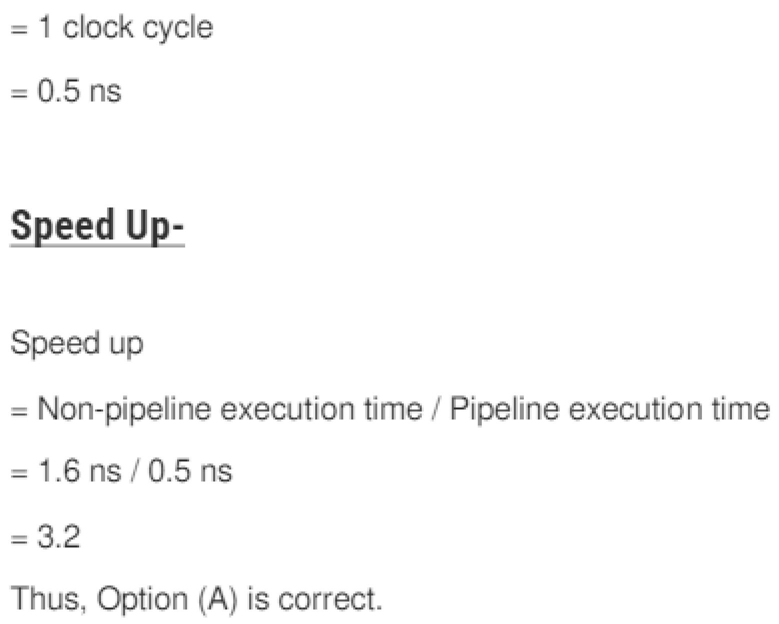
= 1 / frequency

= 1 / (2 gigahertz)

= 1 / (2 x 10^9 hertz)

= 0.5 ns

Pipeline Execution Time- Since there are no stalls in the pipeline, so ideally one instruction is executed per clock cycle. So, Pipeline execution time


= 1 clock cycle

= 0.5 ns

Speed Up-

Speed up

= Non-pipeline execution time / Pipeline execution time

= 1.6 ns / 0.5 ns

= 3.2

Thus, Option (A) is correct.

20. Consider a non-pipelined processor with a clock rate of 2.0 gigahertz and average cycles per instruction of 4.1. The same processor is upgraded to a pipelined processor with five stages but due to the internal pipeline delay, the clock speed is reduced to 4.5 gigahertz. Assume there are no stalls in the pipeline. Find the speed up achieved in this pipelined processor.

SUPERSCALAR PROCESSORS:

20. In Super Scalar processor, find the In-order Execution, In-Order 2 way execution and out of order execution.

For the following instructions.

(1) R1= R2 + 5

(2) R3 = R1 / 3

(3) R5 = R3 * 4

(4) R6 = R4 + R5

(5) R7 = R2 + R3

CACHE MEMORY:

21. A computer has a 256 KB, 4-way set associative, write back data cache with block size of 32 bytes. The processor sends 32 bit addresses to the cache controller. Each cache tag directory entry contains in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit.

Explanation: A set-associative scheme is a hybrid between a fully associative cache, and direct mapped cache. It's considered a reasonable compromise between the complex hardware needed for fully associative caches (which requires parallel searches of all slots), and the simplistic direct-mapped scheme, which may cause collisions of addresses to the same slot (similar to collisions in a hash table).

Number of blocks = Cache-Size/Block-Size = 256 KB / 32 Bytes = 2^{13}
Number of Sets = 2^{13} / 4 = 2^{11}
Tag + Set offset + Byte offset = 32
Tag + 11 + 5 = 32
Tag = 16

22. A block-set associative cache memory consists of 128 blocks divided into four block sets . The main memory consists of 16,384 blocks and each block contains 256 eight bit words.

1. How many bits are required for addressing the main memory?
2. How many bits are needed to represent the TAG, SET and WORD fields?

For main memory, there are 2^{14} blocks and each block size is 2^8 bytes (A byte is an eight-bit word)

1. Size of main memory $= 2^{14} \times 2^8 = 4MB$ ($22 -$ bits required for addressing the main memory).
2. For WORD field, we require $8 -$ bits, as each block contains 2^8 words.

As there are 4 blocks in 1 set, 32 sets will be needed for 128 blocks. Thus SET field requires $5 -$ bits.

Then, TAG field requires $22 - (5+8) = 9 -$ bits

9-bits (for tag)	5- bits (for set)	8-bits (for word)

23. A 4-way set associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. How many number of bits available for the TAG field ?

Cache size = 256 KB = 2^{18} B
Number of tag bits = 6
Memory is represented as:

Tag	Set	Block
6	x	y

Therefore, number of sets = 2^x
Number of blocks = 2^y
Cache size = Number of sets * Number of Lines per set * Block size
$2^{18} = 2^x * 4 * 2^y$
$2^{x+y} = 2^{16}$
x+y = 16 bits
Therefore, width of physical address = 6 + 16 = 22 bits
Memory size = 2^{22} = 4 MB

TLB

24. Suppose that the guest and host machines both use three-level page tables. The host has a hardware-refilled TLB. When running the guest OS, what is the TLB

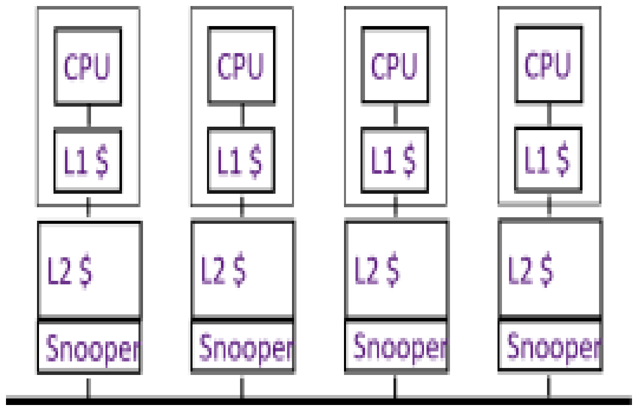
miss latency if the TLB access takes 1 cycle and the memory latency is 50 cycles per access?

1 + 50 + 50 + 50 = 151 cycles A hardware TLB refill involves a page table walk of the shadow page tables in memory, same as if there were no virtualization in effect. Note that the guest page tables are not directly involved. The TLB miss latency must also include the cost of the initial TLB lookup (1 cycle).

Now suppose that the guest machine uses two-level page tables instead, while the host continues to use three-level page tables. When running the guest OS, what is the TLB miss latency if the TLB access takes 1 cycle and the memory latency is 50 cycles per access?

1 + 50 + 50 + 50 = 151 cycles The shadow page table walk is agnostic to the structure of the guest page tables.

25. L2 cache can act as a filter to reduce the amount of L1 coherence traffic in a snoopy cache-coherence protocol. If a coherence request misses in the L2 cache, there is no need to probe the L1 cache for the given line.



(i) Explain how a strictly exclusive L2 cache can also be used to optimize snooping by the L1 cache.

If a coherence request hits in the strictly exclusive L2 cache, the given line cannot be in the L1 cache, so there is no need to probe the L1. Another acceptable answer is that coherence requests to upgrade permission for lines already present in the L1 (e.g., S to M) do not need to be broadcasted to the L2.

(ii) Could a non-inclusive, non-exclusive L2 cache (i.e., neither strictly inclusive nor strictly exclusive) be similarly used to optimize snooping by the L1 cache? Explain.

No, hits and misses at the L2 cache reveal no information about what the L1 cache contains, so the L1 must snoop every coherence request. A "yes" answer is also acceptable if it sufficiently explains how the L2 can track inclusivity with the L1 (e.g., an extra bit per L2 line to indicate it is shared with the L1).

26. Explain whether the following microarchitectural optimizations are permitted under the TSO consistency model.
(i) Can a load from one thread on a multithreaded core bypass a value from the write buffer stored by a different thread on the same core? (Recall that a write buffer holds data from committed stores waiting to be written to the cache.)

No – TSO requires store atomicity (a global memory order for stores), so a thread cannot see the result of another thread's write early.

(ii) Can a write buffer coalesce writes to the same word from the same thread? (Writes from two stores, not necessarily consecutive in program order, are merged into the same write buffer entry.)

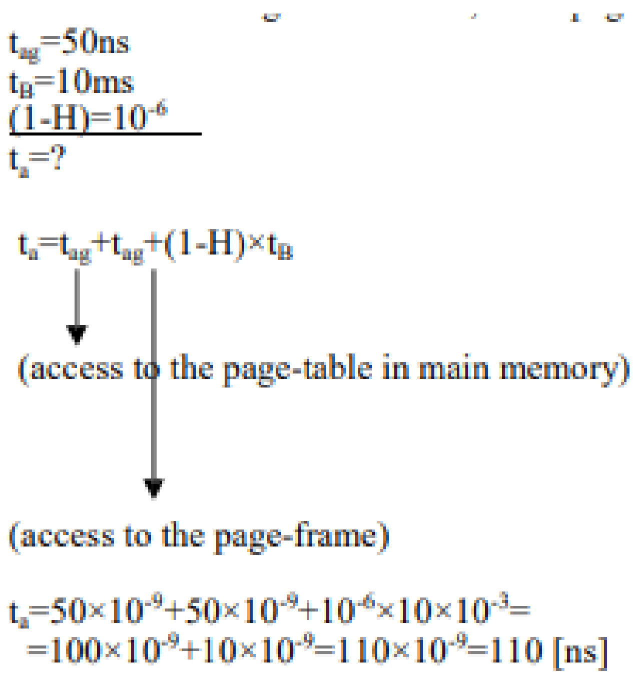
No – TSO requires that stores appear in program order, so the write buffer must remain FIFO. Coalescing two non-consecutive stores creates a store-store reordering.

(iii) Can hardware prefetching be used with the L1 cache for both loads and stores?

Yes – fetching a cache line earlier does not affect consistency so as the loads and stores are performed in the same order without prefetching. The coherence protocol will ensure that any intervening modifications to the line will erase an incorrect prefetch.

<u>VIRTUAL MEMORY</u>:

24. A computer with virtual memory has an access time to main memory 50 ns, the time to transfer a block from the virtual into main memory is 10 ms. The probability for the page-fault is 10^{-6} . What is the average access time, if the page-table is in the main memory?



$t_{ag}=50\text{ns}$
$t_B=10\text{ms}$
$\underline{(1-H)=10^{-6}}$
$t_a=?$

$$t_a = \underbrace{t_{ag}}_{\text{(access to the page-table in main memory)}} + \underbrace{t_{ag}}_{\text{(access to the page-frame)}} + (1-H)\times t_B$$

$$t_a=50\times10^{-9}+50\times10^{-9}+10^{-6}\times10\times10^{-3}=$$
$$=100\times10^{-9}+10\times10^{-9}=110\times10^{-9}=110\ [\text{ns}]$$

25. Computer with virtual memory has the following features:

- length of the virtual address is 38 bits,
- the page size is 16 KB,
- length of the physical address is 32 bits.

a) How many bits is the length of page descriptor, if in addition to the frame number (FN), additional parameters occupy another 6 bits?

b) What is the maximum size of the page-table in bytes?

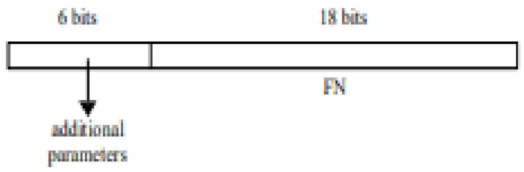
a) How many bits is the length of page descriptor, if in addition to the frame number (FN), additional parameters occupy another 6 bits?

$n=38$
$f=32$
page size = 16 KB$=2^p=2^{14}$B

- a number of pages in virtual memory: $2^{n-p}=2^n\div2^p=2^{38}\div2^{14}=2^{24}$
- a number of page frames in main memory: $2^{f-p}=2^f\div2^p=2^{32}\div2^{14}=2^{18}$ (FN)

Page descriptor = frame number (FN) + 6

Page descriptor = 24 bits = 3 B



b) What is the maximum size of the page-table in bytes?

num_pages x page_descriptor

$2^{24}3$B x $= 2^{20}\times 2^{4}3$B x $= 2^{4}\times 3$MB $= 16\times 3$MB $= 48$MB page-table size in main memory.

26. On a computer with a 32-bit memory address and the length of the memory location of 1 byte is installed set-associative cache. Cache size is 16 KB, block (line) size is 16 Bytes, set associative cache is 4-way.

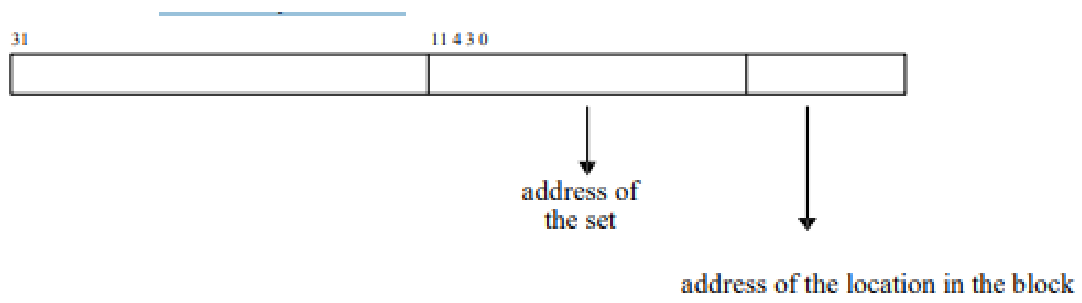
a) How many sets are there in cache?

b) Which bits in the memory address determine the address of the set?

a) How many sets are there in cache?

$n=32$
$M=16KB=2^4KB=2^{14}B$
block$=B=16B=2^4B$
$E=4=2^2$

$M=S\times E\times B$
$S=M\div(E\times B)=2^{14}\div(2^2\times2^4)=2^{14-6}=2^8=256$ sets



b) Address bits of set: 4-11.

27. In a computer with cache, we have the average number of clock periods per instruction equal to 4, if there are no misses in the cache.

a) What is the real number of clock periods per instruction, if the probability of miss in the cache is 10%? For the replacement of the block (line) in the cache, we need 5 clock periods for read and 10 for write accesses. Assume that each instruction requires an average of 2 memory accesses and that 20% of all are write accesses.

$CPI_I=4$
$(1-H)=10\%=0,1$
$M_I=2$
$N_R=5$
$N_W=10$
$P_W=0,2$
$P_R=0,8$

$CPI_R=CPI_I+M_I\times(1-H)\times miss\_penalty$

$miss\_penalty = P_W\times num\_periods\_for\_write + P_R\times num\_periods\_for\_read = P_W\times N_W + P_R\times N_R$

$CPI_R= 4 + 2 \times 0.1 \times (0.2 \times 10 + 0.8 \times 5) = 5.2$

b) What is the real CPI, if we increase the probability of hit to 95%?

$CPI_I=4$
$M_I=2$
$(1\text{-}H)=0{,}05$
$\text{miss\_penalty} \underline{=6}$
$CPI_R=?$

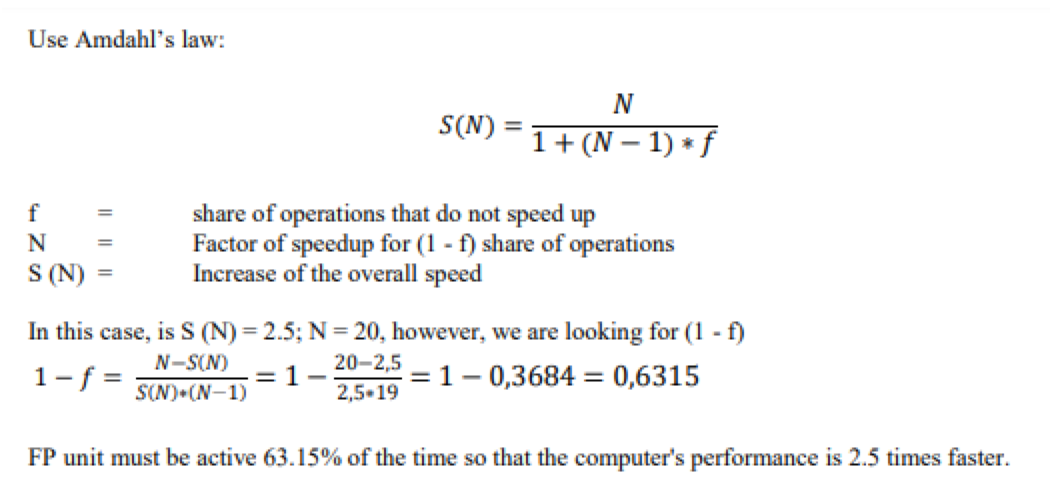
$CPI_R=CPI_I+M_I\times(1\text{-}H)\times \text{miss\_penalty}$
$CPI_R=4+2\times0{,}05\times6=4{,}6$

28. The computer has a main memory access time of 60 ns. We want to reduce this time to 20 ns by adding cache. Determine how fast the cache must be (access time) if we can expect a 90% probability of a hit.

$t_{ag}=60$ ns
$t_a=20$ ns
$H=90\%=0{,}9$
$t_{ap}=?$

$t_a=t_{ap}+(1\text{-}H)\times t_{ag}$
$t_{ap}=t_a\text{-}(1\text{-}H)\times t_{ag}$
$t_{ap}=20\times10^{-9}\ [s] - (1\text{-}0{,}9)\times60\times10^{-9}\ [s]=20\times10^{-9}\text{-}6\times10^{-9}=14\times10^{-9}\ [s]=14\ [ns]$

29. We want to speed up computer performance with an additional unit for calculating in floating point format. This unit is 20 times faster than the same operations without unit. What percentage of a total computer time must this unit be active to achieve an overall increase in computer speed for 2.5 times?



Use Amdahl's law:

$$S(N) = \frac{N}{1 + (N-1) * f}$$

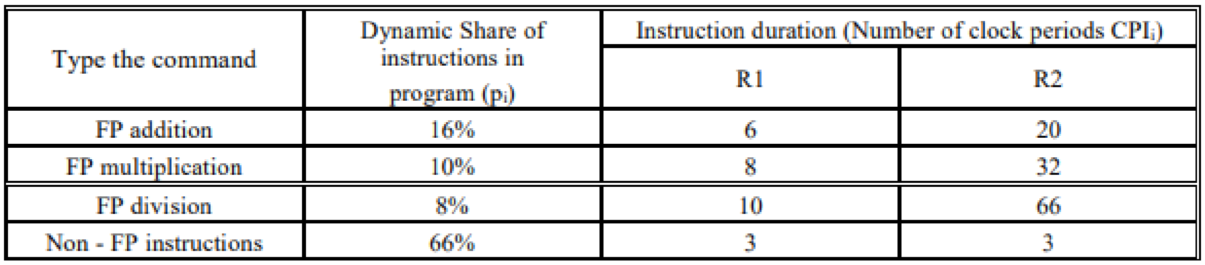
f	=	share of operations that do not speed up
N	=	Factor of speedup for (1 - f) share of operations
S (N)	=	Increase of the overall speed

In this case, is S (N) = 2.5; N = 20, however, we are looking for (1 - f)

$$1 - f = \frac{N - S(N)}{S(N) * (N-1)} = 1 - \frac{20 - 2{,}5}{2{,}5 * 19} = 1 - 0{,}3684 = 0{,}6315$$

FP unit must be active 63.15% of the time so that the computer's performance is 2.5 times faster.

30. We want to compare the computers R1 and R2, which differ that R1 has the machine instructions for the floatingpoint operations, while R2 has not (FP operations are implemented in the software using several non-FP instructions). Both computers have a clock frequency of 400 MHz. In both we perform the same program, which has the following mixture of commands:



Type the command	Dynamic Share of instructions in program (p_i)	Instruction duration (Number of clock periods CPI_i)	
		R1	R2
FP addition	16%	6	20
FP multiplication	10%	8	32
FP division	8%	10	66
Non - FP instructions	66%	3	3

a) Calculate the MIPS for the computers R1 and R2.

b) Calculate the CPU program execution time on the computers R1 and R2, if there are 12000 instructions in the program?

c) At what mixture of instructions in the program will both computers R1 and R2 be equally fast?

a) CPI = $\sum_{i=0}^{3} CPI_i * p_i$

MIPS = $\frac{f_{CPE}}{CPI*10^6}$

Computer R1:

CPI $= \sum_{i=1}^{3} CPI_i * p_i = 0{,}16 * 6 + 0{,}1 * 8 + 0{,}08 * 10 + 0{,}66 * 3 = 4{,}54$

Computer R1 needs an average of 4.54 clock periods for one instruction

MIPS = $\frac{f_{CPE}}{CPI*10^6} = \frac{400*10^6}{4{,}54*10^6} = 88{,}1$

Computer R1 executes an average of 88 100 000 instructions per second.

Computer R2:

CPI $= \sum_{i=1}^{3} CPI_i * p_i = 0{,}16 * 20 + 0{,}1 * 32 + 0{,}08 * 66 + 0{,}66 * 3 = 13{,}66$

Computer R2 needs an average of 13.66 clock periods for one instruction

MIPS = $\frac{f_{CPE}}{\mathrm{CPI}*10^6} = \frac{400*10^6}{13{,}66*10^6} = 29{,}28$

Computer R2 executes an average of 29 280 000 instructions per second.

b) $$\boldsymbol{CPU}_{\text{time}} = \frac{\text{Number\_of\_instructions}}{\boldsymbol{MIPS} * \mathbf{10^6}}$$

Another form of the equation to calculate the CPU time is:

$$CPU_{time} = Number\_of\_instructions * CPI * t_{CPU}$$

Computer R1:

$$\boldsymbol{CPU}_{\text{time}} = \frac{\text{Number\_of\_instructions}}{\boldsymbol{MIPS} * \mathbf{10^6}} = \frac{12000}{88{,}1 * 10^6} = 136{,}2 * 10^{-6} = 136{,}2\ \mu s$$

Computer R2:

$$\boldsymbol{CPU}_{\text{time}} = \frac{\text{Number\_of\_instructions}}{\boldsymbol{MIPS} * \mathbf{10^6}} = \frac{12000}{29{,}28 * 10^6} = 410 * 10^{-6} = 410\ \mu s$$

c) Programs without FP instructions ...