Solution 1

Infix Expression:

$$A - B + C \times (D \times E - F) \div (G + H \times K)$$

Postfix Conversion Steps:

1. $D \times E \rightarrow DE\times$
2. $DE \times -F \rightarrow DE \times F-$
3. $H \times K \rightarrow HK\times$
4. $G + HK\times \rightarrow GHK \times +$
5. $C \times (DE \times F-) \rightarrow CDE \times F - \times$
6. $CDE \times F - \times \div GHK \times + \rightarrow CDE \times F - \times GHK \times +\div$
7. $A - B \rightarrow AB-$
8. Combine: $AB - \ CDE \times F - \times \ GHK \times + \div \ +$

Final Postfix Expression:

$$AB - \ CDE \times F - \times \ GHK \times + \div \ +$$

Solution 2

1. Analyze the hexadecimal value

The hexadecimal value oxCAFEBABE consists of 4 bytes:

- CA(byte 1)
- FE(byte 2)
- BA(byte 3)
- BE(byte 4)

2. Big Endian Storage

In Big Endian format, the most significant byte(MSB) is sorted first. So, oxCAFEBABE is stored as:

- Address 0x100 stores CA
- Address 0x101 stores FE
- Address 0x102 stores BA
- Address 0x103 stores BE

3.Little Endian Storage

- Address 0x100 stores BE
- Address 0x101 stores BA
- Address 0x102 stores FE
- Address 0x103 stores CA

4.Final Answer

As stated above, the final answer should be:

Address	Big Endian	Little Endian
0x100	CA	BE
0x101	FE	BA
0x102	BA	FE
0x103	BE	CA

Solution 3

As known in this problem: The virtual address size is 20 bits, the physical address size is 14 bits, The page size is 1 KB, which is 1024 bytes. We will use the information above to solve the problems below.

a) How many pages exist in virtual and physical memory, and explain your reasons?

The proof is as follows:
1. Virtual Memory Pages:

- Virtual address space: Since the virtual address size is 20 bits, the total virtual addressable space is 2^{20} bytes (or 1048576 bytes).
- Virtual pages: Each page is 1 KB (1024 bytes), so the number of pages in virtual memory is:

$$\begin{aligned}\text{Virtual Pages} &= \frac{2^{20}}{1024} \\ &= 2^{10} = 1024\,\text{pages}.\end{aligned}$$

2. Physical Memory Pages:

- Physical address space: Since the physical address size is 14 bits, the total physical addressable space is 2^{14} bytes (or 16384 bytes).

- Physical pages: Each page is 1 KB (1024 bytes), so the number of pages in physical memory is:

$$\begin{aligned}\text{Physical Pages} &= \frac{2^{14}}{1024} \\ &= 2^4 = 16\,\text{pages}.\end{aligned}$$

b) What would the format of a virtual and physical address be, and explain your reason?

1. Virtual Address Format:

- The virtual address consists of two parts: the page number and the page offset.
- The page offset: Since each page is 1 KB (1024 bytes), we need 10 bits to represent the offset within a page.
- The page number: The remaining 10 bits (since total virtual address size is 20 bits) are used to represent the page number in virtual memory.
- So, the virtual address format is:

$$\text{Virtual Address} = \text{Page Number (10 bits)} + \text{Page Offset (10 bits)}.$$

2. Physical Address Format:

- The physical address also consists of two parts: the physical page number and the page offset.
- The page offset: Since each page is still 1 KB (1024 bytes), the page offset is the same as for virtual memory and requires 10 bits.
- The physical page number: The remaining 4 bits (since total physical address size is 14 bits) are used to represent the physical page number.
- So, the physical address format is:

$$\text{Physical Address} = \text{Physical Page Number (4 bits)} + \text{Page Offset (10 bits)}.$$

c) Suppose a program accesses a virtual address 0xA48F8. Describe how the system tries to access this address

Given Address: 0xA48F8

1. Convert the address to binary

$$0xA48F8 = 1010\,0100\,1000\,1111\,1000_2$$

2.Split the address into page number and page offset:

- **Page Number**: The first 10 bits represent the page number.

$$\text{Page Number} = 1010\,0100\,10$$

- **Page Offset**: The last 10 bits represent the offset within the page.

$$\text{Page Offset} = 0011\,1110\,00$$

3.Translate the page number to a physical page number:
The system uses the page number 1010 0100 10 to look up the page table, which maps the virtual page number to a physical page number.

4.Use the page offset to find the byte within the page:
The page offset 0011 1110 00 specifies the exact byte within the physical page.

Solution 4

As give in the problem: The word size is 16 bits, the instruction size is 16 bits, the number of operations (opcodes) is 25, the number of registers is 12, memory is word addressable

1. Format the instruction The instruction consists of: - Opcode: There are 25 operations, so we need $\lceil \log_2(25) \rceil = 5$ bits for the opcode. - Register address: The machine has 12 registers, so we need $\lceil \log_2(12) \rceil = 4$ bits for the register address. - Memory address: The remaining bits in the instruction will be used for the memory address.

$$\text{Instruction format} = \text{Opcode (5 bits)} + \text{Register Address (4 bits)} + \text{Memory Address (7 bits)}.$$

2. Maximum Memory Capacity
The memory address field has 7 bits, meaning the maximum number of memory locations is:

$$\text{Number of memory locations} = 2^7 = 128 \text{ words}.$$

Since each word is 16 bits (or 2 bytes), the maximum memory capacity is:

$$\text{Maximum memory capacity} = 128 \times 16 \text{ bits} = 2048 \text{ bits} = 256 \text{ bytes}.$$

Thus, the maximum capacity for memory is $\boxed{256 \text{ bytes}}$.

Solution 5

a) What is the size of an address?

Since the memory has 32 units, we need $\lceil \log_2(32) \rceil = 5$ bits to address all the memory units.

Thus, the size of an address is $\boxed{5 \text{ bits}}$.

b) To address a memory unit inside a chip, how many bits are required as an offset?

Each chip has $\frac{32}{4} = 8$ memory units. To address these 8 memory units inside a chip, we need $\lceil \log_2(8) \rceil = 3$ bits as an offset.

Thus, the number of bits required as an offset is $\boxed{3 \text{ bits}}$.

c) Show the place of each memory unit in the memory organization when low-order interleaving is used.

In low-order interleaving, memory units are assigned to chips according to the low-order bits of their address. The addresses will be distributed as follows:

Address	Chip
0, 4, 8, 12, 16, 20, 24, 28	Chip 1
1, 5, 9, 13, 17, 21, 25, 29	Chip 2
2, 6, 10, 14, 18, 22, 26, 30	Chip 3
3, 7, 11, 15, 19, 23, 27, 31	Chip 4

d) Show the place of each memory unit in the memory organization when high-order interleaving is used.

In high-order interleaving, memory units are assigned to chips according to the high-order bits of their address. Each chip handles a contiguous block of memory addresses. The addresses will be distributed as follows:

Address	Chip
0, 1, 2, 3, 4, 5, 6, 7	Chip 1
8, 9, 10, 11, 12, 13, 14, 15	Chip 2
16, 17, 18, 19, 20, 21, 22, 23	Chip 3
24, 25, 26, 27, 28, 29, 30, 31	Chip 4

Solution 6

Assume the numbers are stored at addresses X and Y, and the result should be stored at address Z.

Here is the assembly language program:

```
LOAD X       ; Load the value from address X into the accumulator
STORE TEMP   ; Store it in temporary memory (TEMP)

LOAD Y       ; Load the value from address Y into the accumulator
STORE COUNTER ; Store it in COUNTER (loop counter)

LOAD ZERO    ; Load zero into the accumulator
STORE RESULT ; Initialize the result to 0
```

```
MUL_LOOP:
    LOAD RESULT     ; Load the current result
    ADD TEMP        ; Add TEMP to the current result (accumulation)
    STORE RESULT    ; Store the new result

    LOAD COUNTER    ; Load the counter
    SUB ONE         ; Subtract 1
    STORE COUNTER   ; Store the new counter value

    SKIPCOND 400    ; If the counter is 0, skip the next instruction
    JUMP MUL_LOOP   ; Otherwise, continue the loop

    STORE Z          ; Store the result into address Z
```

Explanation:

- `LOAD X`: Loads the first operand from address X.
- `STORE TEMP`: Stores the first operand in a temporary location (TEMP).
- `LOAD Y`: Loads the second operand from address Y.
- `STORE COUNTER`: Stores the second operand into COUNTER, which is the number of times to repeat the addition.
- `LOAD ZERO` and `STORE RESULT`: Initializes the result to zero.
- `MUL_LOOP`: This is the loop where the multiplication is performed using addition. We add the value in TEMP (the value at address X) to the result, and decrement the counter.
- `SKIPCOND 400`: If the counter is zero, the loop terminates.
- `STORE Z`: Stores the result in address Z.

This program implements multiplication by using repeated addition, where the value at X is added to the accumulator Y times.