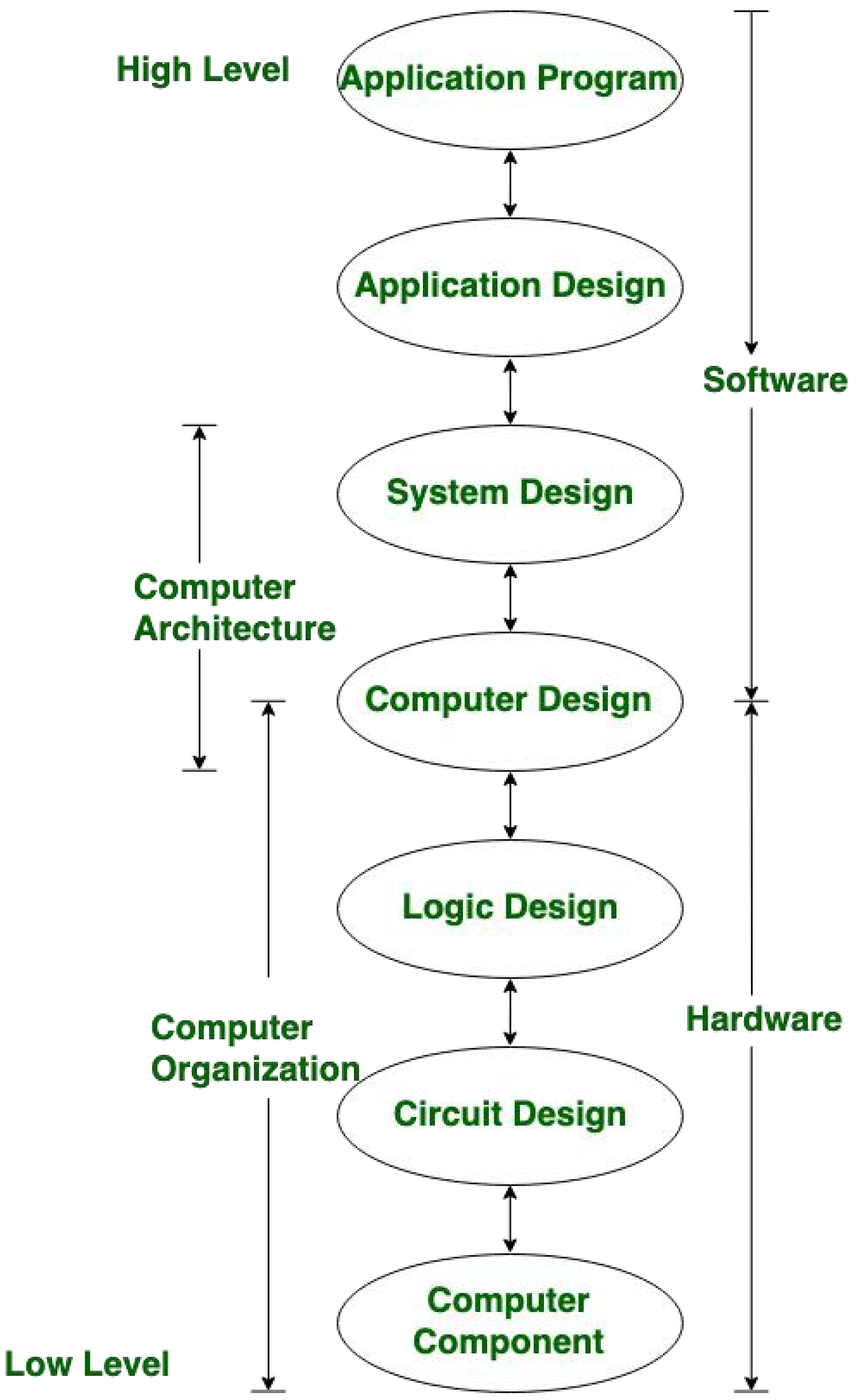
1. CS 2006 COMPUTER ORGANIZATION AND ARCHITECTURE

Computer Architecture is concerned with the way hardware components are connected together to form a computer system. **Computer Organization** is concerned with the structure and behaviour of a computer system as seen by the user. It acts as the interface between hardware and software



Computer System

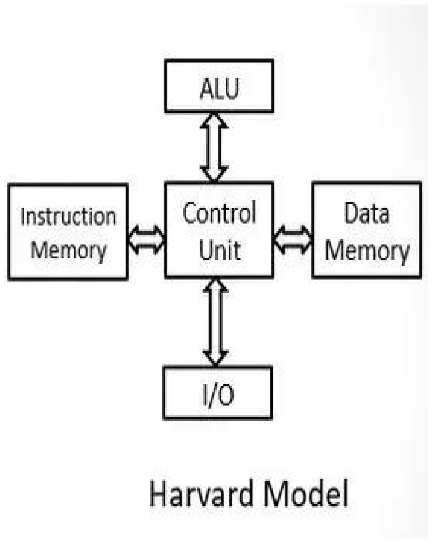
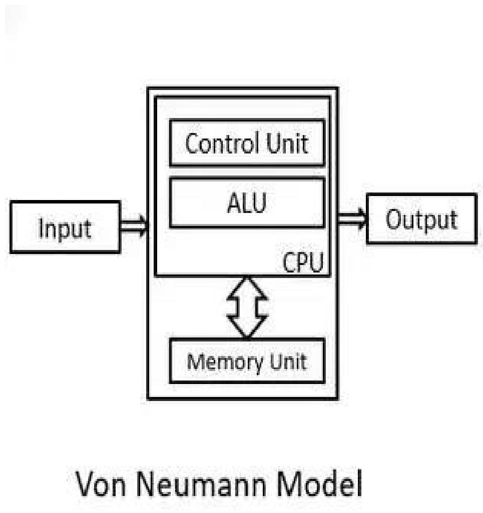
Difference between Computer Architecture and Computer Organization:

S. No.	**Computer Architecture**	**Computer Organization**
1.	Architecture describes what the computer does.	The Organization describes how it does it.
2.	Computer Architecture deals with the functional behavior of computer systems.	Computer Organization deals with a structural relationship.
3.	In the above figure, it’s clear that it deals with high-level design issues.	In the above figure, it’s also clear that it deals with low-level design issues.
4.	Architecture indicates its hardware.	Where Organization indicates its performance.
5.	As a programmer, you can view architecture as a series of instructions, addressing modes, and registers.	The implementation of the architecture is called organization.
6.	For designing a computer, its architecture is fixed first.	For designing a computer, an organization is decided after its architecture.
7.	Computer Architecture is also called Instruction Set Architecture (ISA).	Computer Organization is frequently called microarchitecture.
8.	Computer Architecture comprises logical functions such as instruction sets, registers, data types, and addressing modes.	Computer Organization consists of physical units like circuit designs, peripherals, and adders.
9.	The different architectural categories found in our	CPU organization is classified into three categories based on the

S. No.	**Computer Architecture**	**Computer Organization**
	computer systems are as follows: 1. Von-Neumann Architecture 2. Harvard Architecture 3. Instruction Set Architecture 4. Micro-architecture 5. System Design	number of address fields: 1. Organization of a single Accumulator. 2. Organization of general registers 3. Stack organization
10.	It makes the computer's hardware visible.	It offers details on how well the computer performs.
11.	Architecture coordinates the hardware and software of the system.	Computer Organization handles the segments of the network in a system.
12.	The software developer is aware of it.	It escapes the software programmer's detection.
13.	Examples- Intel and AMD created the x86 processor. Sun Microsystems and others created the SPARC processor. Apple, IBM, and Motorola created the PowerPC.	Organizational qualities include hardware elements that are invisible to the programmer, such as interfacing of computer and peripherals, memory technologies, and control signals.

Difference between Von Neumann and Harvard Architecture

Point of Comparison	Harvard Architecture	Von Neumann Architecture
Arrangement	In Harvard architecture, the CPU is connected with	In Von-Neumann architecture, there is no separate data and

	both the data memory (RAM) and program memory (ROM), separately.	program memory. Instead, a single memory connection is given to the CPU.
	 ALU Instruction Memory Control Unit Data Memory I/O Harvard Model	 Control Unit ALU CPU Input Output Memory Unit Von Neumann Model
Hardware requirements	It requires more hardware since it will be requiring separate data and address bus for each memory.	In contrast to the Harvard architecture, this requires less hardware since only a common memory needs to be reached.
Space requirements	This requires more space.	Von-Neumann Architecture requires less space.
Speed of execution	Speed of execution is faster because the processor fetches data and instructions simultaneously .	Speed of execution is slower since it cannot fetch the data and instructions at the same time.
Space usage	It results in wastage of space since if the space is left in the data memory then the instructions memory cannot use the space of the data memory and vice-versa.	Space is not wasted because the space of the data memory can be utilized by the instructions memory and vice-versa.
Controlling	Controlling becomes complex since data and instructions are to be fetched simultaneously.	Controlling becomes simpler since either data or instructions are to be fetched at a time.

Q.1 Consider a non-pipelined processor operating at 2.5 GHz. It takes 5 clock cycles to complete an instruction. You are going to make a 5-stage pipeline out of this processor. Overheads associated with pipelining force you to operate the pipelined processor at 2 GHz. In a given program, assume that 30% are memory instructions, 60% are ALU instructions and the rest are branch instructions. 5% of the memory instructions cause stalls of 50 clock cycles each due to cache misses and 50% of the branch instructions cause stalls of 2 cycles each. Assume that there are no stalls associated with the execution of ALU instructions. For this program, the speedup achieved by the pipelined processor over the non-pipelined processor (round off to 2 decimal places) is __________ .

Note – This question was Numerical Type.
(A) 2.16
(B) 2.50
(C) 1.50
(D) 1.16

Answer: (A)

Explanation: Assume the total number of instructions to be 'm'.

For a non-pipelined processor:

Given that,
It takes 5 clock cycles to complete an instruction operating at 2.5GHz.
One clock cycle time=1/(2.5*10^9)= 0.4ns
For m instructions total number of clock cycles required= 5m.
Time taken to complete 5m clock cycles= 0.4*5m= 2m ns

For a pipelined processor:

Given that, Pipeline is 5-staged, Overheads associated with pipelining force to operate the pipelined processor at 2 GHz.
One clock cycle time= 1/(2*10^9)= 0.5ns
For m instructions total number of clock cycles required=
0.3m*(0.05*(50+1)+0.95*(1))+0.6m*(1)+0.1m*(0.5*(2+1)+0.5*(1))= 1.85m.
Time taken to complete 1.85m clock cycles= 0.5*1.85m= 0.925m ns

Therefore, Speedup=Time taken without pipelining/Time taken with pipelining
= 2m ns/ 0.925m ns
= 2.16

Q.2 A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter (all values in decimal)?
(A) 400
(B) 500
(C) 600
(D) 700

Answer: (C)

Explanation:

Here, size of instruction = 24/8 = 3 bytes.

Program Counter can shift 3 bytes at a time to jump to next instruction.

So the given options must be divisible by 3. only 600 is satisfied.

The Performance Equation:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clocks}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock}}$$

The performance equation analyzes execution time as a product of three factors that are relatively independent of each other.

This equation remains valid if the time units are changed on both sides of the equation. The left-hand side and the factors on the right-hand side are discussed in the following sections.

The three factors are, in order, known as the instruction count (IC), clocks per instruction (CPI), and clock time (CT). CPI is computed as an effective value.

Performance Equation

The performance equation analyzes execution time as a product of three factors that are relatively independent of each other.

Instruction Count

Computer architects can reduce the instruction count by adding more powerful instructions to the instruction set. However, this can increase either CPI or clock time, or both.

Clocks Per Instruction

Computer architects can reduce CPI by exploiting more instruction-level parallelism. If they add more complex instructions it often increases CPI.

Clock Time

Clock time depends on transistor speed and the complexity of the work done in a single clock. Clock time can be reduced when transistor sizes decrease. However, power consumption increases when clock time is reduced. This increase the amount of heat generated.

Instruction Count

Instruction (IC) count is a dynamic measure: the total number of instruction executions involved in a program. It is dominated by repetitive operations such as loops and recursions.

Instruction count is affected by the power of the instruction set. Different instruction sets may do different amounts of work in a single instruction. CISC processor instructions can often accomplish as much as two or three RISC processor instructions. Some CISC processor instructions have built-in looping so that they can accomplish as much as several hundred RISC instruction executions.

For predicting the effects of incremental changes, architects use execution traces of benchmark programs to get instruction counts. If the incremental change does not change the instruction set then the instruction count normally does not change. If there are small changes in the instruction set then trace information can be used to estimate the change in the instruction count.

For comparison purposes, two machines with different instruction sets can be compared based on compilations of the same high-level language code on the two machines.

Clocks Per Instruction

Clocks per instruction (CPI) is an effective average. It is averaged over all of the instruction executions in a program.

CPI is affected by instruction-level parallelism and by instruction complexity. Without instruction-level parallelism, simple instructions usually take 4 or more cycles to execute. Instructions that execute loops take at least one clock per loop iteration. **Pipelining (overlapping execution of instructions)** can bring the average for simple instructions down to near 1 clock per instruction. **Superscalar pipelining (issuing multiple instructions per cycle)** can bring the average down to a fraction of a clock per instruction.

For computing clocks per instruction as an effective average, the cases are categories of instructions, such as branches, loads, and stores. Frequencies for the categories can be extracted from execution traces. Knowledge of how the architecture handles each category yields the clocks per instruction for that category.

Clock Time

Clock time (CT) is the period of the clock that synchronizes the circuits in a processor. It is the reciprocal of the clock frequency.

For example, a 1 GHz processor has a cycle time of 1.0 ns and a 4 GHz processor has a cycle time of 0.25 ns.

Clock time is affected by circuit technology and the complexity of the work done in a single clock. Logic gates do not operate instantly. A gate has a propagation delay that depends on the number of inputs to the gate (fan in) and the number of other inputs connected to the gate's output (fan out). Increasing either the fan in or the fan out slows down the propagation time. Cycle time is set to be the worst-case total propagation time through gates that produce a signal required in the next cycle. The worst-case total propagation time occurs along one or more signal paths through the circuitry. These paths are called critical paths.

In the integrated circuit technology has been greatly affected by a scaling equation that tells how individual transistor dimensions should be altered as the overall dimensions are decreased. The scaling equations predict an increase in speed and a decrease in power consumption per transistor with decreasing size. Technology has improved so that about every 3 years, linear dimensions have decreased by a factor of 2. Transistor power consumption has decreased by a similar factor. Speed increased by a similar factor until about 2005. At that time, power consumption reached the point where air cooling was not sufficient to keep processors cool if the ran at the highest possible clock speed.

Problem Statement

Suppose a program (or a program task) takes 1 billion instructions to execute on a processor running at 2 GHz. Suppose also that 50% of the instructions execute in 3 clock cycles, 30% execute in 4 clock cycles, and 20% execute in 5 clock cycles. What is the execution time for the program or task?

We have the instruction count: 10^9 instructions. The clock time can be computed quickly from the clock rate to be 0.5×10^{-9} seconds. So we only need to to compute clocks per instruction as an effective value:

Value	**Frequency**	**Product**
3	0.5	1.5
4	0.3	1.2
5	0.2	1.0
CPI =		3.7

Then we have

Execution time = $1.0\times10^9 \times 3.7 \times 0.5\times10^{-9}$ sec = 1.85 sec.

Chapter-2

A computer has 64 bit instruction and 12 bit address. If there are 250 three address instruction and 525 two address instruction, how many one address instructions are possible?

Solution-I:

So total instruction size =64 bits. In 3 address instructions , we have 3 address fields each having 12 bits, so No of bits for opcode remaining = 64 - 36 = 28 bits.

28 bit	12 bit	12 bit	12 bit

no of opcode possible 2^{28}

So no of operations possible = 2^{28}

But we have 250 3 address instructions , so

No of opcodes unused and can be used for 2 address instructions = $2^{28} - 250$

We know :

No of 2 address instructions possible = No of unused(free) opcodes from 3 address instruction * $2^{\text{Address field size}}$

Here 2Address field size as one address field is appended to opcode of 2 address instruction from 3 address instruction , and this is done for every free opcode..

Hence no of 2 address instructions (opcode) possible = $(2^{28} - 250) * 2^{12}$

But we have 525 such instructions as mentioned in the question..

Hence no of unused opcodes = $[(2^{28} - 250) * 2^{12} - 525]$

This can be used for 1 address instruction generation..

So no of one address instructions possible = No of unused opcodes in 2 address instruction format * $2^{\text{Addr field size}}$

Answer = $[(2^{28} - 250) * 2^{12} - 525] * 2^{12}$

Registers R1 and R2 of a computer contain the decimal values 1200 and 4600, we have to find effective adress of associated memory operand in each instruction:

Load 20(R1),R5 : This means load 20+R1 into R5 . R1= 1200 , R1 + 20 = 1220 , so R5 have 1220 , Effective address of R5 is 1220.

Move #3000,R5 : This means move value 3000 into R5 , so effective address is part of the instruction whose value is 3000.

Now R5 = 3000

Store R5,30(R1,R2) : This means 30+R1+R2 and store the result into R5 .

so R5 = 30+1200+4600 = 5830 , so now R5 value is 5830 , the effective address is 5830.

Add -(R2),R5 : This means -1 from R2 value and store the result into R5 . So R5= 4600 - 1 = 4599 , effective address of R5 is 4599 . It is pre decrement addressing.

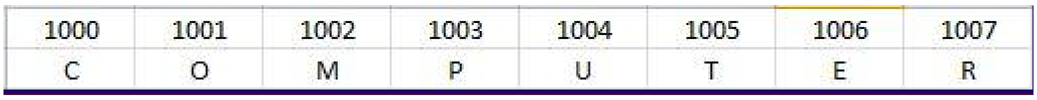
Subtract (R1)+,R5 : This means effective address is contents of R1 so EA = 1200 .

It is post increment addressing .

Chapter2 Homework

2.2 Consider a computer has a byte-addressable memory organized in 32-bit words according to the big-endian scheme. A program reads ASCII characters entered at a keyboard and stores them in successive byte locations, starting at location 1000. Show the contents of the two memory words at locations 1000 and 1004 after the word “Computer” has been entered.

Values correspond to the character below it.



1000	1001	1002	1003	1004	1005	1006	1007
C	O	M	P	U	T	E	R

2.4 Registers R4 and R5 contain the decimal numbers 2000 and 3000 before each of the following addressing modes is used to access the memory operand. What is the effect address (EA) in each case?

a. 12(R4) which is 12 + R4 so EA = 12 + 2000 = 2012

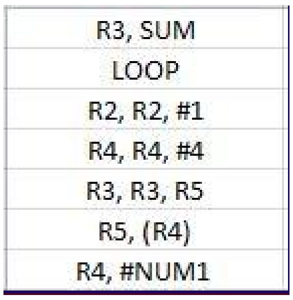
b. (R4, R5) which is R4 + R5 so EA = 2000 + 3000 = 5000

c. 28(R4, R5) which is 28 + R4 + R5 so EA = 28 + 2000 + 3000 = 5028

d. (R4)+ which is (2000)+ so EA = 2000 + 1 = 2001

e. –(R4) which is –(2000) so EA = 2000 – 1 = 1999

2.9 Rewrite the addition loop in figure 2.8 so that the numbers in the list are accessed in the reverse order; that is, the first number accessed is the last one in the list, and the last number accessed is at memory location NUM1.



R3, SUM
LOOP
R2, R2, #1
R4, R4, #4
R3, R3, R5
R5, (R4)
R4, #NUM1

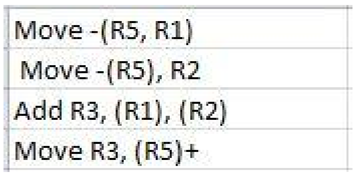
Advertisements

REPORT THIS AD

This Loop seems to execute much faster and more efficiently compared to the loop in figure 2.8.

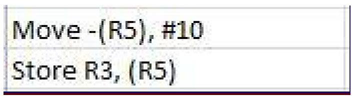
2.19 Register R5 is used in a program to point to the top of a stack containing 32-bit numbers. Write a sequence of instructions using the Index, Autoincrement, and Autodecrement addressing modes to perform each of the following tasks (assuming ten or more elements in each stack):

a. Pop the top two items off the stack, add them, then push the result onto the stack



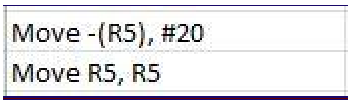
```
Move -(R5, R1)
 Move -(R5), R2
Add R3, (R1), (R2)
Move R3, (R5)+
```

b. Copy the fifth item from the top into register R3



```
Move -(R5), #10
Store R3, (R5)
```

c. Remove the top ten items from the stack



```
Move -(R5), #20
Move R5, R5
```

Registers R4 and R5 contain numbers 2,000 and 3,000 before the following addressing modes are used to access a memory operand. what is the effective address (EA) in each case?

a.) 12(R4). EA = 12 +(R4) = 12 + 2,000 = 2,012.

b.) (R4, R5). EA = (R4) + (R5) = 2,000 + 3,000 = 5,000.

c.) 28(R4, R5). EA = 28 + ((R4) + (R5)) = 28 + 5,000 = 5,028.

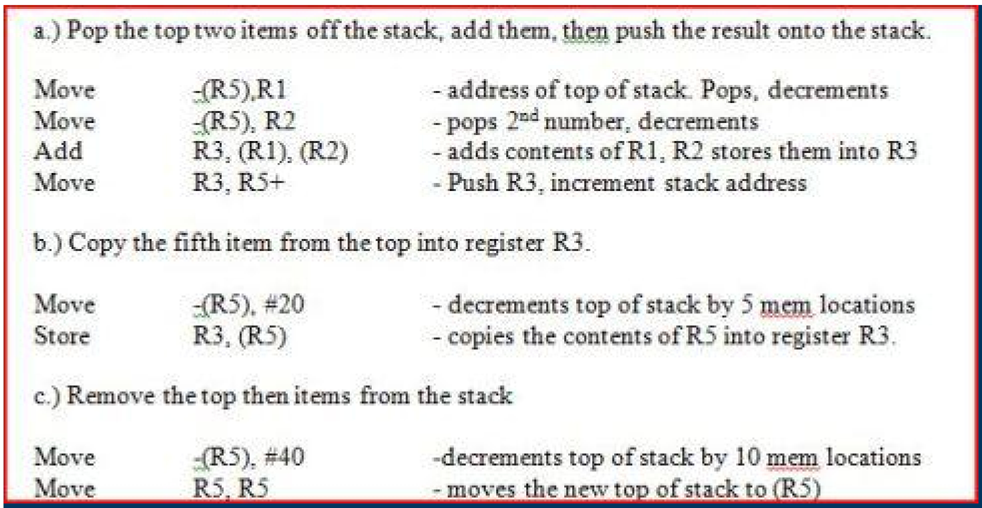
d.) (R4)+. EA = would be 2,000, because we are using post-auto increment.

Fetch operand from memory (2,000) , then R4 <- R4 + 1.

e.) -(R4). EA = would be 1,999, because we are using pre-auto increment.

R4 <- R4 – 1, then fetch operand from memory (1,999).

R5 is used in a program to store the address of the top of the stack, use auto-increment and auto-decrement addressing modes to perform each of the following tasks.



a.) Pop the top two items off the stack, add them, then push the result onto the stack.

```
Move        -(R5),R1            - address of top of stack. Pops, decrements
Move        -(R5), R2           - pops 2nd number, decrements
Add         R3, (R1), (R2)      - adds contents of R1, R2 stores them into R3
Move        R3, R5+             - Push R3, increment stack address
```

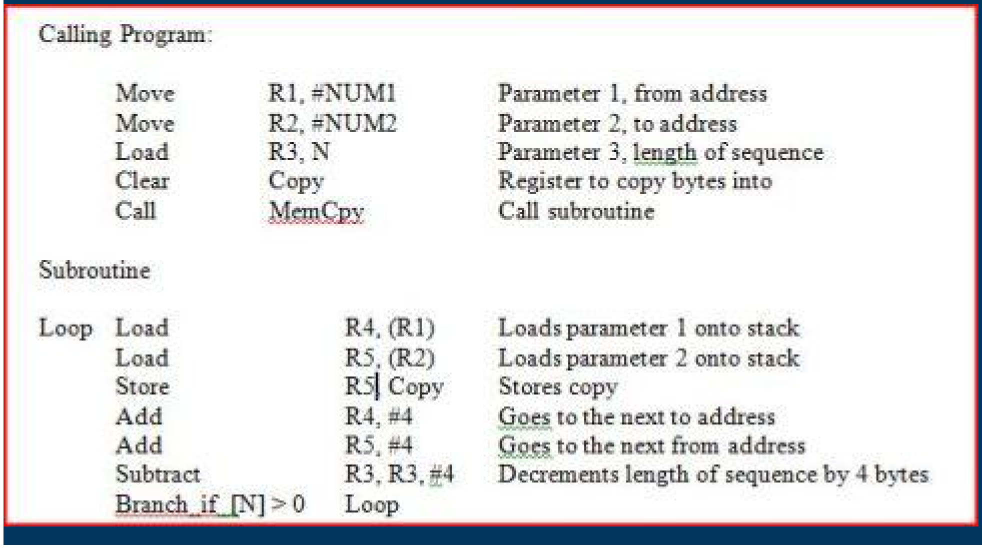
b.) Copy the fifth item from the top into register R3.

```
Move        -(R5), #20          - decrements top of stack by 5 mem locations
Store       R3, (R5)            - copies the contents of R5 into register R3.
```

c.) Remove the top then items from the stack

```
Move        -(R5), #40          -decrements top of stack by 10 mem locations
Move        R5, R5              - moves the new top of stack to (R5)
```

2.27) Write a MemCpy subroutine, to copy a sequence of bytes in one area of memory to another. The subroutine accepts 3 input parameters: from address, to address, length of bytes. Subroutine should copy bytes in the order of increasing addresses.



```
Calling Program:

        Move        R1, #NUM1       Parameter 1, from address
        Move        R2, #NUM2       Parameter 2, to address
        Load        R3, N           Parameter 3, length of sequence
        Clear       Copy            Register to copy bytes into
        Call        MemCpy          Call subroutine

Subroutine

Loop    Load                R4, (R1)        Loads parameter 1 onto stack
        Load                R5, (R2)        Loads parameter 2 onto stack
        Store               R5, Copy        Stores copy
        Add                 R4, #4          Goes to the next to address
        Add                 R5, #4          Goes to the next from address
        Subtract            R3, R3, #4      Decrements length of sequence by 4 bytes
        Branch_if_[N]>0     Loop
```

Explain the significance of carry and overflow flag.

In computer processors, the overflow flag (sometimes called V flag) is usually a single bit in a system status register used to indicate when an arithmetic overflow has occurred in an operation, indicating that the signed two's-complement result would not fit in the number of bits used for the operation (the ALU width)

A computer has 64-bit instructions and 12 bit addresses. If there are 352 three-address

instructions, and 2256 no of two-address instructions then how many one-address
instructions can be formulated?

Maximum Possible 3 address instructions are:228.
Maximum possible 2 address instructions are: (228 – 352) X 212
So, Maximum possible 1 address instructions are: ((228 – 352) X 212-2256) X 212

Write the number of memory references required for executing the following instructions:

i) ADD R1,(R2)+

ii) SUB #10,R2

iii) MOV R1, 20(R3,R4)

iv) AND R1,R2

v) Increment A

i) ADD R1,(R2)+ 3 memory references
(1 for instruction + 1 for R2 + 1 for store result after Incrementing R2 value)
ii) SUB #10,R2 1 memory references (1 for instruction)
iii) MOV R1, 20(R3,R4) 2 memory references
(1 for instruction +1 for move value to memory Address [20 + R3 + R4])
iv) AND R1,R2 1 memory references (1 for instruction)
v) Increment A 3 memory references (1 for instruction + 1 for accessing A +1 for
writing the new value into A)