ASSIGNMENT -AUGUST 29

**********TWO ADDRESS BASED ISA(INSTRUCTION SET ARCHITECTURE)*******

Now consider the following assembly code

```
ADD  $s0,  $s0,  $s1
ADD  $s2,  $s2,  $s3
MUL  $s0,  $s0,  $s2
ADD  $s5,  $s0,  $zero
```

Clearly this is same as MIPS assembly code , and in each step we used only two registers and the output is obtained in $s5.This way the problem can be done.But the problem is the values of the variables are changed and for furthur use it be might be inefficient to access them .But this can be over come by saving them into the temporary variables and working on them to obtain the result.

**********ONE ADDRESS BASED ISA(INSTRUCTION SET ARCHITECTURE)*******

Now consider the following assembly code,

```
lw      $s0
add     $s1
sw      $s0
lw      $s2
add     $s3
sw      $s2
lw      $s1
mul     $s2
sw      $s5
```

-->LW - LOADWORD

Here load word loads the value in register $s0.In general it loads the value into given register.

-->ADD - ADDITION

Here add instruction adds the given content into the immediate loaded content .Here $s1 is added to $s0.

-->MUL – MULTIPLICATION

Here mul instruction multiplies the given content into the immediate loaded content .Here $s2 is multiplied to $s1 ,which is loaded previously

-->SW – STORE WORD

Here store word stores the value in register $s5 of the resultant .In general it stores the value into given register.

Now with the above instruction set now each step uses only single register and gives us the resultant .By this assembly code format and predefined rules we can achieve as desired.

**********ZERO ADDRESS BASED ISA(INSTRUCTION SET ARCHITECTURE)******

Now consider the following assembly code,

```
PUSH    $S0
PUSH    $S1
ADD
PUSH    $S2
PUSH    $S3
ADD
MUL
POP     $S5
```

Here in this architecture ,we have a predefined method called stack , which takes the values and puts them in stack and some predefined functions which when called performs the desied operations on stack.

-- →>PUSH

Here push instruction puts the element into the stack .So here in the first step $S0 goes to the empty stack

-- →>ADD

Here add function call adds the elements in the stack taken two at a time and puts the resultant in the stack as a single entity.

So here ADD performed after pushing $S0 and $S1 adds them and puts them in stack.

-- →>MUL

Here mul function call multiplies the elements in the stack taken two at a time and puts the resultant in the stack as a single entity.

So here MUL performed after adding multiplies them and puts them in stack.

-- →>POP

Pop function call pops the resultant entity in the stack into the given register $S5.

In this way the above ISA can be performed.