BITS PILANI, DUBAI CAMPUS II Semester 2018 – 2019 III Yr. C.S.

Course: CS F363 Compiler Construction

**Individual** LAB EXERCISE: Total Marks: 10 Weightage: **10 %**

Date for Demonstration and Record Submission:

**On or Before 16 April** Tuesday Batch: Dr AAJ

Record should contain **Source Code, Input** and **Output**. Record can be submitted **only after** successful demonstration.

Note: Delayed Submission will result in reduction in marks.

Copying not permissible. Issue Date: 12/04/19

**The students are required to work in their individual linux accounts in the linux server. The first statement in your program should contain your IDNO as a comment line.**

Write an Algorithm and Implement the Algorithm using C/C++/JAVA (any programming language of your choice) for Generation of 80X86 CODE using QUADRAPLES (intermediate code ) as input. Assume one register only **%eax**.

Input : (read from a text file) quadruples Output: 80X86 Code

Test Case 1:

-----------------------------------------------------------------------

**Input:** (vi quad.in)

= 20 nil a

= 30 nil b

= 40 nil c

= a nil eax

- b eax eax

- c eax eax

= eax nil z

------------------------------------------------------------------------

The quadraples above corresponds to the following block.

a=20;

b=30;

c=40;

z=a-b-c;

----------

**Output (quad.s :** to be generated by your program**)**

**--------------------------------------------------------------------**

**.global main**

**msg:**

**.string "Final result of LHS var = %d\n"**

**.comm a,4**

**.comm b,4**

**.comm c,4**

**.comm z,4**

**main:**

**pushl %ebp**

**movl %esp, %ebp**

*movl $20, a*

*movl $30, b*

*movl $40, c*

*movl a, %eax*

*subl b, %eax*

*subl c, %eax*

*movl %eax, z*

**pushl z**

**pushl $msg**

**call printf**

**leave**

**ret**

-----------------------------------------------------------------

The highlighted text (bold) indicates the compulsory and fixed part of your program.

The italicized text is the variable part and it depends on the no of quadruple instructions and operands. Assume that the **maximum number** of operands on the **right hand side** of the assignment statement is **6**.

**You need to Execute** the generated 80x86 assembly language code generated by you through your program (stored in the file: quad.s).

verify that your code is working correctly and giving correct output results using the following steps:

$ **gcc quad.s**

$**a.out**

**Final result of LHS var = -50**

**$**

---------------------------------------------------------------------------

**Your program should work for any number of operands (say 1,2,3,4,5,6) on the RHS.**

**Meaning of some of the terms in the above assembly code.**

**.global <symbol>** : The symbol is external variable

**msg :** The msg which you want to print at the end.

**.comm <symbol>, <size>** : .comm a, 4 (Variable a is 4 bytes long integer)

pushl, movl, addl, subl are 80x86 Assembly Lang. instructions.

 In x86, the instruction pointer is the register %eip and the stack pointer is in the register  %esp[2](http://codearcana.com/posts/2013/05/21/a-brief-introduction-to-x86-calling-conventions.html#fn:4)

The base pointer is conventionally used to mark the start of a function's stack frame, or the area of the stack managed by that function. Local variables are stored below the base pointer and above the stack pointer. The start of each function has a preamble saves the old base pointer and initializes a new one and the end of each function has epilogue that restores the old base pointer:

**my\_function:**

push %ebp *# Preamble: save the old %ebp.*

movl %esp, %ebp *# Point %ebp to the saved %ebp and the new stack frame.*

---------------------------------------------------------------------------------------------------------------

* %esp points to the last thing pushed on the stack.
* %eip points to the next thing to execute.
* call <addr> pushes the current value of %eip and changes %eip to <addr>.
* ret pops the next value off the stack into %eip.
* Arguments are pushed onto the stack before a function call.