Department of Computer Science
Ashoka University

Programming Language Design and Implementation: CS1319

Assignment Type: Individual

Assignment 1: `Introduction to PLDI` *Marks: 100*
Assign Date: *Sep 10, 2023* Submit Date : *23:55, Sep 16, 2023*

You must submit your assignment using the naming convention `FirstName_LastName_Assignment1.pdf`.

---

1. Compare and contrast the following languages **[8 * 5 = 40]**

   (a) C++
   (b) Dlang
   (c) Haskell
   (d) Java
   (e) Prolog
   (f) Perl
   (g) Python
   (h) SQL

   with references to :

   i. Paradigm of computation (Imperative, Declarative, Object-Oriented, Logic, Meta-Programming, Functional),
   ii. Time and space efficiency of generated code,
   iii. Portability across target processors, operating systems, device form factors, etc.,
   iv. Developers' productivity to code, test, and fix bugs, and
   v. Typical application areas.

2. Consider the following C program (numbers on the left mark the line numbers and are not part of the program). **[10 + 50 = 60]**

```c
#include <stdio.h>
const int n1 = 25;
const int n2 = 39;

int main() {
    int num1, num2, diff;

    num1 = n1;
    num2 = n2;
    diff = num1 - num2;

    if (num1 - num2 < 0)
        diff = -diff;

    printf("\nThe absoute difference is: %d", diff);

    return 0;
}
```

(a) Explain the phases of a multi-pass compiler using above the program. [10]

(b) The above program is compiled by MS-VC to generate an assembly file[1].
The assembly-code file contains various declarations including translated assembly instructions. For ease of understanding, the source lines have also been interspersed (and no optimization is allowed). Note that in this assembly file semicolon (;) starts a C++ style comment that continues till the end of the line.

```
; Listing generated by Microsoft (R) Optimizing Compiler Version 18.00.21005.1
    .686P
    .XMM
    include listing.inc
    .model flat

INCLUDELIB MSVCRTD
INCLUDELIB OLDNAMES

PUBLIC _n1
PUBLIC _n2
CONST SEGMENT
_n1 DD 019H
_n2 DD 027H
CONST ENDS
PUBLIC _main
PUBLIC ??_C@_0BP@CMAHBJAF@?6The?5absoute?5difference?5is?3?5?$CFd?$AA@ ; `string'
EXTRN __imp__printf:PROC
EXTRN __RTC_CheckEsp:PROC
EXTRN __RTC_InitBase:PROC
EXTRN __RTC_Shutdown:PROC
; COMDAT rtc$TMZ
rtc$TMZ SEGMENT
__RTC_Shutdown.rtc$TMZ DD FLAT:__RTC_Shutdown
rtc$TMZ ENDS
; COMDAT rtc$IMZ
rtc$IMZ SEGMENT
__RTC_InitBase.rtc$IMZ DD FLAT:__RTC_InitBase
rtc$IMZ ENDS
; COMDAT ??_C@_0BP@CMAHBJAF@?6The?5absoute?5difference?5is?3?5?$CFd?$AA@

CONST SEGMENT
??_C@_0BP@CMAHBJAF@?6The?5absoute?5difference?5is?3?5?$CFd?$AA@ DB 0aH, 'T'
DB 'he absoute difference is: %d', 00H ; `string'
CONST ENDS

; Function compile flags: /Odtp /RTCsu /ZI
; COMDAT _main
_TEXT SEGMENT
_diff$ = -32 ; size = 4
_num2$ = -20 ; size = 4
_num1$ = -8 ; size = 4
_main PROC ; COMDAT
```

[1]Refer to x86/64 assembly language and related information at: Introduction to x64 Assembly, Lecture 03: x86 instruction set or x86 Assembly Guide , and go no further.

```
; 5 : int main() {
push ebp
mov ebp, esp
sub esp, 228 ; 000000e4H
push ebx
push esi
push edi
lea edi, DWORD PTR [ebp-228]
mov ecx, 57 ; 00000039H
mov eax, -858993460 ; ccccccccH
rep stosd

; 6 : int num1, num2, diff;
; 7 :
; 8 : num1 = n1;
mov eax, DWORD PTR _n1
mov DWORD PTR _num1$[ebp], eax

; 9 : num2 = n2;
mov eax, DWORD PTR _n2
mov DWORD PTR _num2$[ebp], eax

; 10 : diff = num1 - num2;
mov eax, DWORD PTR _num1$[ebp]
sub eax, DWORD PTR _num2$[ebp]
mov DWORD PTR _diff$[ebp], eax

; 11 :
; 12 : if (num1 - num2 < 0)
mov eax, DWORD PTR _num1$[ebp]
sub eax, DWORD PTR _num2$[ebp]
jns SHORT $LN1@main

; 13 : diff = -diff;
mov eax, DWORD PTR _diff$[ebp]
neg eax
mov DWORD PTR _diff$[ebp], eax
$LN1@main:

; 14 :
; 15 : printf("\nThe absoute difference is: %d", diff);
mov esi, esp
mov eax, DWORD PTR _diff$[ebp]
push eax
push OFFSET ??_C@_0BP@CMAHBJAF@?6The?5absoute?5difference?5is?3?5?$CFd?$AA@
call DWORD PTR __imp__printf
add esp, 8
cmp esi, esp
call __RTC_CheckEsp

; 16 :
; 17 : return 0;
xor eax, eax
; 18 : }
```

```
98  pop edi
99  pop esi
100 pop ebx
101 add esp, 228 ; 000000e4H
102 cmp ebp, esp
103 call __RTC_CheckEsp
104 mov esp, ebp
105 pop ebp
106 ret 0
107 _main ENDP
108 _TEXT ENDS
109
110 END
```

Annotate each line of the `CONST` segment (`CONST SEGMENT` to `CONST ENDS`) and `TEXT` segment (`TEXT SEGMENT` to `TEXT ENDS`) of the assembly file to explain the functionality of the instruction and the connection to the original C program. Naturally, ignore all commented source lines for annotation. [**50**]