Name: KEY **Id#**

COE 205, Term 092

Computer Organization & Assembly Programming

Quiz# 6

Date: Monday, May 17, 2010

Q1. Compare macros and procedures in terms of parameter passing, types of parameters, invocation mechanism, memory space, execution time and assembly time.

1. **Parameter passing**: Parameter passing in a macro invocation is similar to that in a procedure call of a high-level language. The arguments are listed as part of a macro call. Parameter passing in a procedure call often involves the stack. The number of stack operations in preparation for a procedure call grows in direct proportion to the number of parameters passed. This, in addition to the call/ret overhead, increases the overhead and affects the performance. Macros avoid this overhead by text substitution but increase the space requirement.
2. **Types of parameters**: Since a macro is a text substitution mechanism, a variety of parameter types can be passed. For example, the opcode of an instruction could be passed as a parameter. Procedures do not have such flexibility in parameter passing.
3. **Invocation mechanism**: Macro invocation is done at assembly time by text substitution. However, procedure invocation is done at run time by transferring control to the procedure. This leads to the following tradeoff. Macros tend to increase the length of the executable code due to macro expansions. This leads to increased assembly time.
   In summary, the tradeoffs are that using macros results in faster execution of the code. However, macros result in increased memory space due to macro expansions. Procedures save space, as only one copy of the procedure is kept. However, procedure invocation overhead (to pass parameters via the stack and for call/ret) increases the execution time. Note that macro invocation causes assembly-time overhead but not run-time overhead.

Q2. Give an example where it is better to use macros than procedures.

Macros are useful in defining macro-instructions that extend the instruction set of a processor. Macros are also useful when text substitution is the only way available. For example, suppose that we want to preserve the content of registers ECX, EDX, ESI and EDI across procedure calls. We can conveniently do this by the following two macros:

```
save_regs MACRO                restore_regs  MACRO
             PUSH ECX                           POP EDI
             PUSH EDX                           POP ESI
             PUSH ESI                           POP EDX
             PUSH EDI                           POP ECX
          ENDM                               ENDM
```

It is not possible to write a procedure to do the same.

Q3. Given the following macros and macro invocations, determine the assembly code generated by the assembler:

(i) GET_BIG FIRST, SECOND

```
GET_BIG      MACRO  WORD1, WORD2
                      LOCAL EXIT
                      MOV EAX, WORD1
                      CMP EAX, WORD2
                      JG  EXIT
                      MOV EAX, WORD2
                 EXIT:
               ENDM
```

MOV EAX, FIRST
CMP EAX, SECOND
JG ??0000
MOV EAX, SECOND
??0000:

(ii) A LABEL DWORD

```
      BLOCK 5
   BLOCK  MACRO  N
            K=1
            REPT N
              DWORD K
              K=2*K+1
            ENDM
          ENDM
```

A LABEL DWORD
DWORD 1
DWORD 3
DWORD 7
DWORD 15
DWORD 31

(i) SAVE_REGS < EAX, EBX, ECX>

```
SAVE_REGS  MACRO  REGS
              IRP D, <REGS>
                  PUSH D
              ENDM
            ENDM
```

PUSH EAX
PUSH EBX
PUSH ECX