CS 271 Computer Architecture and Assembly Language
Final Exam

- *Calculator and* 8.5 x 11 *note page permitted.*
- *Multiple-Choice problems,* 2 *pts each. Others as marked.*
- *For all* MASM *questions,* **WriteDec** *is a procedure that displays the contents of the* **eax** *register as an integer.* **CrLf** *is a procedure that moves the cursor to the beginning of the next line.*

1. (5 pts) What is the correct order of steps in the instruction execution cycle?

   2 Update the instruction pointer to point to the next instruction

   4 Fetch the instruction operand(s) if necessary

   5 Execute the instruction

   1 Fetch the instruction at the instruction pointer into the instruction register

   3 Decode the instruction in the instruction register

   6 Go back to the beginning of the instruction execution cycle

2. (3 pts) In code fragment *R*, suppose that variables *y* and *z* are declared as *DWORD*, and *z* contains a non-negative value. Assuming that the value of *z* is small enough so that no overflow occurs, what is calculated and stored in memory location *y*? (Your answer should be an algebraic expression, not a literal value.)

   3^z

```
;code fragment R
      mov   eax,1
      mov   ebx,3
      mov   ecx,z
      cmp   ecx,0
      je    store
top:
      mul   ebx
      loop  top
store:
      mov   y,eax
```

Use the following declarations and address information for problems # 3 - 4

```
MAXSIZE = 100
.data
list            DWORD       MAXSIZE DUP (?)
```

Recall that *DWORD* specifies 4 bytes (32 bits). After the "program" is loaded and relocated, the label *list* is equivalent to absolute address 1000h. I.E., during program execution, the *list* array starts at memory address 1000h. All addresses are given in hexadecimal; other values are in decimal.

3. (2 pts) How many bytes of memory are reserved by the declaration of *list*? **400**

4. **C** After the statement `mov edi,OFFSET list` is executed, which of the following correctly assigns the value 42 to the 10th element of the *list* array?
   - **A.** `mov list[edi+36],42`
   - **B.** `mov list[9],42`
   - **C.** `mov [edi+36],42`
   - **D.** all of the above
   - **E.** none of the above

5. (3 pts) After code fragment *S* is executed, what are the contents of the first 6 elements of the *list* array.

100	99	98	97	96	95

```
;code fragment S
      mov   edi,OFFSET list
      mov   eax,MAXSIZE
top:
      cmp   eax,0
      jle   quit
      mov   [edi],eax
      dec   eax
      add   edi,TYPE list
      jmp   top
quit:
```

6. (3 pts) After code fragment *T* is executed, what are the contents of the first 6 elements of the *list* array.

1	3	5	7	9	11

```
;code fragment T
      mov   edi,0
      mov   eax,1
      mov   ecx,MAXSIZE
top:
      mov   list[edi],eax
      add   eax,2
      add   edi,4
      loop  top
```

7. (3 pts) After code fragment *U* is executed, what are the contents of the first 6 elements of the *list* array.

1	2	3	4	5	6

```
;code fragment U
      mov   edi,OFFSET list
      mov   ebx,0
      mov   eax,1
      mov   ecx,MAXSIZE
top:
      mov   [edi+ebx],eax
      inc   eax
      add   ebx,TYPE list
      loop  top
```

8. **A** Pipelining is a form of instruction-level parallelism that is implemented by running multiple instruction execution cycles so that
   A. sequential instructions can be in different stages of the cycle at the same time
   B. sequential instructions can be executed simultaneously
   C. each instruction can be processed by multiple CPUs to verify accuracy
   D. an n-stage pipeline can guarantee that a program will run in T/n time (where T is the time to run the program without pipelining).
   E. none of the above

9. **D** Synchronization of parallel processes may be required when
   A. a single program is running on a multi-processor or a multi-computer system
   B. two or more programs are running simultaneously on a single processor
   C. a program implements an algorithm that uses threads
   D. all of the above
   E. none of the above

10. **D** The speed of processor-level parallelism as implemented in hardware depends on
   I. the CPU speed of individual processors
   II. the I/O speed of individual processors
   III. the speed of the interconnection network
   IV. scalability of the parallel architecture
   A. I, II, III **B.** II, III, IV **C.** I, III, IV **D.** I, II, III, IV
   E. none of A - D is correct

11. **D** The speed of processor-level parallelism as implemented in software depends on the
   I. parallelizability of algorithms
   II. features of parallel application programming languages
   III. parallel operating system software
   IV. efficiency of parallel system libraries
   A. I, II, III **B.** II, III, IV **C.** I, III, IV **D.** I, II, III, IV
   E. none of A - D is correct

In problems # 12 - 16, **A**, **B**, *and* **C** *are Boolean expressions. Select* **True** *if the statement is a Boolean identity,* **False** *otherwise.*

12. **A** $1 \cdot A = A$
   A. True **B.** False

13. **B** $\overline{A \cdot B} = \overline{A} \cdot \overline{B}$
   A. True **B.** False

14. **B** $\overline{A + B} = \overline{A} \oplus \overline{B}$
   A. True **B.** False

15. **A** $\overline{A} + \overline{B} = \overline{A \cdot B}$
   A. True **B.** False

16. **A** $A + (B \cdot C) = (A + B)(A + C)$

A. True **B.** False

Use this information to answer questions # 17 - 18.

An algorithm takes 20 seconds to execute on a single 2.4G processor. 40% of the algorithm is sequential. Assuming zero latency and perfect parallelism in the remaining code …

17. (3 pts) Approximately how long should the algorithm take on a parallel machine made of 8 2.4G processors?

9.5 sec.

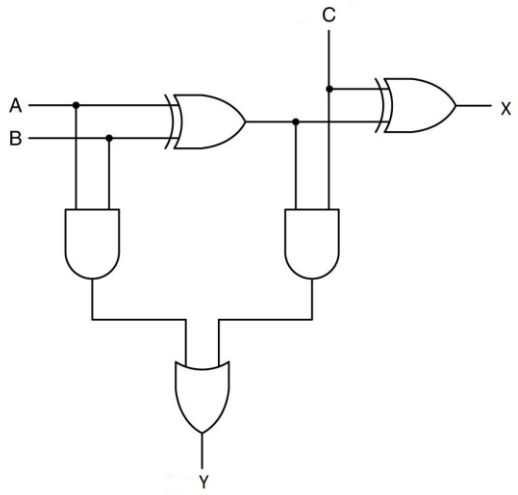
18. (3 pts) Suppose that we can add (with perfect scalability) any number of 2.4 G processors to the system. What is the fastest time that can be achieved?

8 sec.

Use the circuit diagram at the right to answer question # 19.

19. **B** What is the name of this circuit?

A. half adder **B.** full adder
C. 1-bit ALU **D.** multiplexer
E. none of the above



Use this information and the code at the right to answer questions # 20 - 21.

Given the following declarations for an IA-32 processor:

```
MAXSIZE = 10
.data
string BYTE   MAXSIZE DUP (?)
val    DWORD  ?
```

The *ReadString* procedure accepts the address of the memory destination in *edx*, and the maximum number of characters to read in *ecx*. *ReadString* stores the user's input characters in the memory destination, and the actual number of characters in *eax*. The ASCII code for character '0' is 48. Suppose that the user enters the string "5738" (without the quotes) when the *ReadString* procedure is called.

```
;code fragment V
    mov  edx,OFFSET string
    mov  ecx,MAXSIZE
    dec  ecx
    call ReadString
    mov  ecx,eax  ;number of
                  ; digits entered
    mov  val,0    ;initialize val
    mov  esi,OFFSET string
    cld
top:
    mov  eax,0
    lodsb
    sub  eax,48
    mov  ebx,eax
    mov  eax,val
    mov  edx,10
    mul  edx
    add  eax,ebx
    mov  val,eax
    loop top
done:
    mov  eax,val
    call WriteDec
```

20. (3 pts) What is displayed?

5738

21. **C** What is stored in memory at *val*?

A. `0x5 0x7 0x3 0x8`

B. `0x8 0x3 0x7 0x5`

C. `0x6A 0x16 0x0 0x0`

D. `0x0 0x0 0x16 0x6A`

E. none of the above

Use this information and the code at the right to answer questions # 22 - 23.
Space *is a macro that displays a specified number of blank spaces.*

```
;code fragment W
        mov     eax,1
        mov     ebx,0
top:
        call    WriteDec
        Space   2
        cmp     eax,13
        jg      quit
        mov     ecx,ebx
        mov     ebx,eax
        add     eax,ecx
        jmp     top
quit:
```

22. (3 pts) What is the output of MASM code fragment W?

1 1 2 3 5 8 13 21

23. **D** *WriteChar* displays the ASCII character in the *AL* register. Which of the following correctly implements the *Space* macro?

A.
```
Space MACRO x
      push  EAX
      push  ECX
      mov   AL, ' '
      mov   ECX,x
again:
      call  WriteChar
      loop  again

      pop   ECX
      pop   EAX
ENDM
```

B.
```
Space MACRO x
      LOCAL again
      push  ECX
      push  EAX
      mov   AL, ' '
      mov   ECX,x
again:
      call  WriteChar
      loop  again
      pop   EAX
      pop   ECX
ENDM
```

C.
```
Space MACRO x
      LOCAL again
      push  EAX
      push  ECX
      mov   AL, ' '
      mov   ECX,x
again:
      call  WriteChar
      loop  again

      pop   ECX
      pop   EAX
ENDM
```

D. B and C

E. none of these

Use this information to answer questions # 24 - 25.
Suppose that a program's data and executable code require 3200 bytes of memory. A new section of code must be added; it will be used with various values 20 times during the execution of a program. When implemented as a macro, the macro code requires 60 bytes of memory. When implemented as a procedure, the procedure code requires 192 bytes (including parameter-passing, etc.), and each procedure *call* requires 5 bytes.

24. (3 pts) How many bytes of memory will the entire program require if the new code is added as a macro?

4400

25. (3 pts) How many bytes of memory will the entire program require if the new code is added as a procedure?

3492

Use this information to answer questions # 26 - 31.

Given the following MASM data segment declarations:

```
.data
list      DWORD       50 DUP(?)
matrix    DWORD       20 DUP(5 DUP(?))
value     DWORD       10597059          ; decimal
string    BYTE        "Computer Architecture",0
```

The base address of ***list*** is **1000H**

26. (2 pts) What is the hexadecimal base address of *matrix*?

0x10C8

27. (2 pts) What is the hexadecimal address of **list[5*TYPE list]** ?

0x1014

28. (2 pts) In a high-level language, the 27th element of ***list*** is referenced as ***list***[26]. What is the hexadecimal address of ***list***[26] ?

0x1068

29. (2 pts) In a high-level language, the 4th column of the 6th row of ***matrix*** is referenced as ***matrix***[5][3]. What is the hexadecimal address of ***matrix***[5][3] ?

0x1138

30. (2 pts) What is the character in **BYTE PTR string+6** ?

'e'

31. (2 pts) What is the decimal value in **BYTE PTR value+2** ?

161 (177 gets 1 point)

32. (3 pts) Show the postfix (RPN) form of the infix expression **a - (b + c) / (d * e)**

abc+de*/-

33. (3 pts) Find the integer value of the postfix (RPN) expression

2 7 + 3 2 - + 5 / 2 *

4

34. (3 pts) Show the infix form of the postfix (RPN) expression **a b c d + - e / ***

a * (b – (c + d)) / e

Assume that *a*, *b*, *c*, *d*, *e*, and *z* have all been declared as **`REAL10`**. Use the IA-32 FPU instructions in this table:

Instruction	Meaning
`finit`	Initialize the FPU
`fld` *var*	Push value of *var* onto the register stack
`fstp` *var*	Pop value from register stack into *var*
`fadd`	Pop two values from register stack, add, and push result onto register stack
`fsub`	Pop two values from register stack, subtract first popped from second popped, and push result onto register stack
`fmul`	Pop two values from register stack, multiply, and push result onto register stack

35. **C** (3 pts) Which of the following correctly implements the assignment statement:

```
z = a * (b + c) + (d - e)
```

NOTE: Don't forget operator precedence !!

A.
```
finit
fld    a
fmul
fld    b
fadd
fld    c
fadd
fld    d
fsub
fld    e
fstp   z
```

B.
```
finit
fld    b
fld    c
fadd
fld    a
fmul
fld    d
fld    e
fsub
fadd
fstp   z
```

C.
```
finit
fld    a
fld    b
fld    c
fadd
fmul
fld    d
fld    e
fsub
fadd
fstp   z
```

D.
```
finit
fld    a
fld    b
fld    c
fadd
fld    d
fld    e
fsub
fadd
fmul
fstp   z
```

E. none of these

36. (5 pts) The code below uses the *Space* macro defined previously. What output is generated by the MASM "program"?

```
main        PROC
      push  1
      push  1
      push  5
      call  rfinal
      exit
main  ENDP

rfinal      PROC
      push  ebp
      mov   ebp,esp
      mov   eax,[ebp+16]
      mov   ebx,[ebp+12]
      mov   ecx,[ebp+8]
      mul   ebx
      mov   [ebp+16],eax
      cmp   ebx,ecx
      jge   unwind

      inc   ebx
      push  eax
      push  ebx
      push  ecx
      call  rfinal
unwind:
      mov   eax,[ebp+16]
      call  WriteDec
      Space 2
      pop   ebp
      ret   12
rfinal            ENDP
```

120__24__6__2__1