

88

FINAL EXAM

INSTRUCTIONS: Write your name on the back of the last sheet of this test. Read each question carefully and answer them all.

1. (30 points) Name each of the 8086 cpu registers we discussed in this course and explain the significance or use of each one's contents. (NOTE: ax, cx, etc. are NOT names)

General Registers

- AX - ACCUMULATOR - used for arithmetic operations
- BP - BASE REGISTER - used for stack pointer
- CX - COUNT REGISTER - used for loop counter
- SI - SOURCE INDEX REGISTER - used for source pointer
- DI - DESTINATION INDEX REGISTER - used for destination pointer
- SP - STACK POINTER - used for stack pointer
- IP - INSTRUCTION POINTER - used for instruction pointer

Segment Registers

- CS - CODE SEGMENT - contains instruction pointer
- DS - DATA SEGMENT - contains data pointer
- SS - STACK SEGMENT - contains stack pointer
- ES - EXTRA SEGMENT - used for extra segment
- FS - FLAG SEGMENT - used for flag register
- GS - GLOBALLY ACCESSIBLE SEGMENT - used for global segment

Control Registers

- CR0 - CONTROL REGISTER 0 - contains control bits
- CR2 - CONTROL REGISTER 2 - contains address of page fault
- CR3 - CONTROL REGISTER 3 - contains base address of CR3
- CR4 - CONTROL REGISTER 4 - contains control bits
- CR8 - CONTROL REGISTER 8 - contains control bits

Instruction Pointer

IP - INSTRUCTION POINTER - contains address of next instruction

flags? (2)

2. (20 points) Write the 8086 assembly statements necessary to perform each of the following assignment statements. Memory storage definitions are shown below. The arithmetic operators used have their standard meanings. You may assume that the contents of any register may be modified, but ONLY the contents of memory locations A and This may be changed. Be sure to use a data item size consistent with the storage declarations.

```

A      dw  ?
C      dw  ?
THIS  db  ?
IS     db  ?
THE   db  ?
END   db  ?

```

a> $A = 21 * (C + 7)$

```

MOV AX, C
ADD AX, 7
MOV AX, 21
MUL AX
MOV A, AX

```

-2

b> $THIS = (IS / THE) + END$

~~MOV AX, IS
DIV THE
ADD AX, END
MOV THIS, AX~~

Handwritten notes:
 DIVISION HAS A QUOTIENT AND A REMAINDER
 THE REMAINDER IS IN EDI
 DIVISION HAS A MAXIMUM OF 16 BYTES OF DIVISOR
 DIVISION HAS A MAXIMUM OF 32 BYTES OF DIVIDEND

-2

```

MOV AX, THE
DIV AX, IS
ADD AX, END
MOV THIS, AX

```

3. (20 points) For the 8086 assembly language EXTERNAL procedure listed below, write a COMMENT that explains the **intent** of the line or section indicated. That is, explain WHY the line/section is there, not HOW it's done. For example, **push ax** : HOW = pushes ax onto stack, WHY = saves caller's ax register.

```

; a> parameters passed to the procedure are stored in the stack
parameter Character equ word ptr [bp+6] ; b>
set up Result equ word ptr [bp+4] ; c>
; d>
_TEXT SEGMENT word public 'code'
assume cs: _TEXT
public Translate ; d> (expected to)

Translate PROC
    Push bp ; e> set up bp to point to parameters
    mov bp, sp
    push cx ; f>
    push dx ; g> saves caller's registers

; h>
    mov ax, Character
    cmp al, '[' ; i>
    je Brack ; j> IF (Character = '[')
    cmp al, '{' ; k> IF (Character = '{')
    je Brace
    cmp al, '('
    je Paren
    mov Result, 0 ; l> SINCE NO VALUES, THE RESULT IS 0, AND WE CAN RETURN
    jmp Return

; m>
Brack:
    mov Result, 11
    jmp Return

; n>
Brace:
    mov Result, 21
    jmp Return

; o>
Paren:
    mov Result, 31

Return:
    pop dx ; p> RESTORE CALLER'S REGISTER (to esp)
    pop cx ; q> RESTORE CALLER'S REGISTER
    pop bp ; r> RESTORE CALLER'S REGISTER
    ret ; s> END OF PROCEDURE TO THE CALLING PROGRAM
; t> MARKS THE END OF SUBPROGRAM'S CODE
    Restore IP
Translate ENDP
_TEXT ENDS

```

4. (10 points) Provide FIVE points to compare/contrast macros and procedures.

all
code
is
in
memory
at
runtime

CODE OF CODE
- MACRO INSTRUCTIONS CALLS ARE REPLACED BY THEIR CONTENT IN THE CODE AT ASSEMBLY TIME, IT ONLY INCREASES THE LENGTH OF THE ACTUAL PROGRAM.
THE DANGER OF USING MACROS RESULTING FROM THIS APPROACH IS THAT THE INCREASED LENGTH OF CODE CAN TRIGGER ERRORS IN REGARDS TO JUMPING INSTRUCTIONS, WHICH MAY BECOME TOO SHORT.

SPEED OF EXECUTION

USAGE OF PROCEDURES CREATES OVERHEAD, BECAUSE OF THE RESOURCES USED WHEN THEY ARE EXECUTED (AS COMPARED TO CALLING CALLER'S CODE, PASSING PARAMETERS). AS A RESULT OF THIS, USING MACROS PROVIDES FASTER CODE

WHILE CALLING OF A MACRO MIGHT REPLACE A PORTION OF THE CODE TO BE ASSEMBLED AS A RESULT OF THE MACRO DEFINITION, BECAUSE THE MACRO DEFINITION IS ENTERED ON TEXT SUBSTITUTION). PROCEDURES CAN BE MODIFIED AND REUSED.

PROCEDURES CAN BE INCLUDED IN THE PROGRAM, OR STORED IN SEPARATE FILES, AND INCLUDED TOGETHER. THIS PROVIDES FOR REUSE OF PROCEDURES IN CODE USED BY MULTIPLE PROGRAMS.

USE OF MACROS FOR THE PART OF THE PROGRAM AT THE TIME BECAUSE IT REQUIRES CODE LOCATIONS THAT APPEAR TO BE THE SAME TO THE USER.

Macro code placed into program
procedures code not

So do procedures!

5. (20 points) Write two assembly language questions you would expect a student in this class to know the answers to. Provide the answer to each question. (NOTE: Consider depth of knowledge vs breadth of knowledge, topics covered, your expectations of this class, etc.) (BIGGER NOTE: *CHEAP* questions and answers are NOT acceptable!! - after all they do count 10 points each!)

5.1) Write assembly code that will sum up the ⁵⁰⁰⁰⁰ elements of an array of 100 elements, each with a value between -20 and +20, and store the sum in a variable TOTAL.

```

- DATA SEGMENT WORD PUBLIC
    ARRAY DB 100 dup(0)
- DATA TOTAL DW 0
ENDS

```

```

MOV CX, 100
SUB SI, SI
SUB AX, AX
FORN1:
    CMP ARRAY[SI], 0
    JLE NEXT
    MOV BX, ARRAY[SI]
    ADD AX, BX
NEXT:
    ADD SI, 1
    LOOP FORN1
MOV TOTAL, AX

```

5.2) Explain the concept of the interrupt and relate it to your experience with assembly coding.

An interrupt is a signal to the CPU that an event has occurred requiring its immediate attention. When an interrupt occurs, the processor stops the execution of its current job, saves the state of the processor, and transfers control to the program that handles the interrupt. The program that handles the interrupt is called the interrupt service routine (ISR). The ISR is related to the interrupt, and it handles the interrupt. After the ISR has finished, the processor restores the state of the stopped program and resumes its execution. If such a situation occurs on the hardware,

There are two types of interrupts - external and internal. External interrupts originate by I/O devices, the system timer, or occur if there is some hardware failure. Internal interrupts are those which are part of the executed program, and occur for trapping errors. These errors are logical errors such as division by zero (reference page 11).

