Mohammad Hashemi | 97243073
Instructor's Name: Dr. Dara Rahmati
March 9, 2021

HW02 Report

Microprocessor and Assembly Language Course - Winter 2021

1.

```
;-----------------------------------------

; A PROGRAM TO CALCULATE THE MULTIPICATION OF TWO 32-BIT NUMBERS

;-----------------------------------------

            .MODEL SMALL
            .STACK 64

            .DATA
A           DW   5678H, 1234H          ; A = 12345678
B           DW   1111H, 1111H          ; B = 11111111
C           DW   0, 0, 0, 0

            .CODE
MAIN        PROC FAR
            MOV  AX, @DATA
            MOV  DS, AX
            MOV  SI, OFFSET A

            MOV  AX, A[SI]             ; load the lower 16bits of A into AX
            MUL  B                     ; load the lower 16bits of B and multiply by AX
            MOV  C, AX
            MOV  C+2, DX
```

```
        MOV  AX, A+2            ; load the higher 16bits of A into AX
        MUL  B                  ; load the lower 16bits of B and multiply by AX
        ADD  C+2, AX
        ADC  C+4, DX
        ADC  C+6, 0

        MOV  AX, A              ; load the lower 16bits of A into AX
        MUL  B+2                ; load the higher 16bits of B and multiply by AX
        ADD  C+2, AX
        ADC  C+4, DX
        ADC  C+6, 0

        MOV  AX, A+2            ; load the higher 16bits of A into AX
        MUL  B+2                ; load the higher 16bits of B and multiply by AX
        ADD  C+4, AX
        ADC  C+6, DX

        MOV  AH, 4CH
        INT  21H
MAIN    ENDP
        END  MAIN
```

2.

```
;-----------------------------------------

; A PROGRAM TO STORE THE FIRST n PRIME NUMBERS IN NA ARRAY

; NOTE: To change the value of the n, simply change the value of CX
; NOTE: The maximum prime numbers which we can save can be change via ARR capacity in data
segment

;-----------------------------------------
            .MODEL SMALL
            .STACK 64

            .DATA
ARR         DB   20 DUP(?)      ;define an array to store the prime numbers

            .CODE
MAIN        PROC FAR
            MOV  AX, @DATA
            MOV  DS, AX

            MOV  DL, 01H        ; DL = 1
            MOV  CX, 02H        ; CX = n : number of prime numbers
            LEA  DI, ARR        ; DI indicates the first element of the array

NEXT:       MOV  BL, 02         ; BL = 2 as the first denominator
            INC  DL             ; DL += 1
            CMP  DL, 02H        ; DL = 2 as the first prime number
            JE   STORE
            CMP  DL, 03H        ; DL = 3 as the second prime number
            JE   STORE
            CMP  DL, 04H
            JGE  DIVISION

DIVISION:   MOV  AH, 00         ; clear the AH
            MOV  AL, DL
            DIV  BL             ; AL = AL / BL
            CMP  AH, 00         ; check whether the remainder is zero or not
            JE   NEXT           ; if it is zero, then jump back to L1
```

```
            JE   NEXT           ; if it is zero, then jump back to L1
            ADD  BL, 01H        ; BL += 1
            CMP  BL, AL         ; check whether the denominator = AL or not
            JLE  DIVISION       ; if yes then jump back to DIVISION
            JMP  STORE

STORE:      MOV  [DI], DL       ; insert the prime number into the ARR
            INC  DI             ; DI += 1
            LOOP NEXT

            MOV  AH, 4CH
            INT  21H
MAIN        ENDP
            END  MAIN
```

This program stores the first n prime numbers. So n is the input of the program. You can change the value of n by changing the value of CX. As we know CX is a word and the maximum possible value for it is 0FFFFH. But we should notice that the values of prime numbers are too large for higher values of n. The result are stored in ARR array. The type can be DW for larger values of prime numbers. The program has three main parts: 1. NEXT 2. DIVISION 3. STORE

1. NEXT: Here we have a loop for increasing the numbers.
2. DIVISION: In this part, we recognize whether a number is prime or not.
3. STORE: We will store each prime number into the existing array in the data segment.

3.a.

```
 ;-----------------------------------------

; A PROGRAM TO CONVERT BCD TO HEX AND THEN CALCULATE A REMAINDER OF A TWO 16-BIT DIVISION

; NOTE: The length of the first number should be set into N.

;-----------------------------------------
            .MODEL SMALL
            .STACK 64

            .DATA
N           DW   4                ; define the length of the nominator
NUM1        DB   1, 2, 3, 4         ; define nominator as a BCD number into an array
NUM2        DW   0003H              ; define denominator
REM         DW   ?

            .CODE
MAIN        PROC FAR
            MOV  AX, @DATA
            MOV  DS, AX

            ; make SI to point to the least significant digit in NUM1
            MOV  SI, OFFSET NUM1
            ADD  SI, N
            DEC  SI

            MOV  BX, 0              ; BX = 0
            MOV  BP, 1              ; multiple of 10 to multiply every digit

REPEAT:     MOV  AL, [SI]           ; the current digit of the NUM1 to process
            MOV  AH, 0              ; clear AH -> AX = AL
            MUL  BP                 ; AX *= BP
            ADD  BX, AX             ; add result to BX

            MOV  AX, BP             ; AX = BP
```

```
            MOV  AX, BP             ; AX = BP
            MOV  BP, 10
            MUL  BP                 ; AX *= 10
            MOV  BP, AX             ; new multiple of 10

            DEC  SI
            CMP  SI, OFFSET NUM1    ; check whether the string is finished or not
            JGE REPEAT



            MOV  AX, BX
            SUB  DX, DX
            DIV  NUM2               ; AX / NUM2   -> AH = remainder
            MOV  REM, DX            ; store the remainder of the division into REM

            MOV  AH, 4CH
            INT  21H
MAIN        ENDP
```

In this program we want to do a division between two numbers but the first one is in BCD and the second one is in 16 bit HEX format. So firstly we should convert the BCD number to a HEX number. The two inputs are defined as NUM1 and NUM2. NUM1 is an array of bytes which each of the cells indicates a decimal digit.

For the conversion, we iterate the array and build the decimal number from the digits.

After that we have two 16bit numbers and then we do a word-word division which is taught in the source book.

3.b.

```
;-----------------------------------------

; A PROGRAM TO CONVERT STRING TO NUMBER AND THEN CALCULATE A REMAINDER OF A DIVISION

; NOTE: The length of the string should be set into N.

;-----------------------------------------
            .MODEL SMALL
            .STACK 64

            .DATA
N           DW   4                ; define the length of the nominator
NUM1        DW   "1255"             ; define nominator as a string
NUM2        DW   0004H              ; define denominator
REM         DW   ?

            .CODE
MAIN        PROC FAR
            MOV  AX, @DATA
            MOV  DS, AX

            ; make SI to point to the least significant digit in NUM1
            MOV  SI, OFFSET NUM1
            ADD  SI, N
            DEC  SI
                                  ; BX = 0
            MOV  BX, 0            ; multiple of 10 to multiply every digit
            MOV  BP, 1

REPEAT:     MOV  AL, [SI]         ; the current character of the NUM1 to process
            SUB  AL, 48           ; convert ascii character to digit
            MOV  AH, 0            ; clear AH -> AX = AL
            MUL  BP               ; AX *= BP
            ADD  BX, AX           ; add result to BX
```

```
            MOV  AX, BP             ; AX = BP
            MOV  BP, 10
            MUL  BP                 ; AX *= 10
            MOV  BP, AX             ; new multiple of 10

            DEC  SI
            CMP  SI, OFFSET NUM1    ; check whether the string is finished or not
            JGE REPEAT



            MOV  AX, BX
            SUB  DX, DX
            DIV  NUM2               ; AX / NUM2   -> AH = remainder
            MOV  REM, DX            ; store the remainder of the division into REM

            MOV  AH, 4CH
            INT  21H
MAIN        ENDP



            END  MAIN
```

This is a similar program to the previous problem. The only difference is that the first operand of the division operation is stored in a string. So we will have a similar procedure to convert the string into it's hex value.

4.

```
;-----------------------------------------

; A PROGRAM TO SUM OVER ALL THE ODD NUMBERS OF A SERIES

; NOTE: The input serie is stored in an array of numbers
; NOTE: The initial value of CX should be the same as the serie's length
;-----------------------------------------
            .MODEL SMALL
            .STACK 64

            .DATA
ARR         DB   12H, 23H, 0ABH, 01H    ; define a series of numbers
SUM         DW   0000H                  ; put the final result in SUM

            .CODE
MAIN        PROC FAR
            MOV  AX, @DATA
            MOV  DS, AX

            MOV  SI, 00H
            MOV  CX, 04H                ; CX = n : number of prime numbers
            MOV  DH, 00H;               ; DH = 0  for summing

NEXT:       MOV  BX, OFFSET ARR
            MOV  AL, [BX+SI]            ; AL now is the next number in the serie

            MOV  DL, 02                 ; DL = 2
            SUB  AH, AH                 ; AH = 0   (AH will be the remainder of the divion by 2)
            DIV  DL                     ; AL / DL  --> AL will be the quotient of the division
            CMP  AH, 00                 ; check whether the remainder is zero or not
            JE   SKIP                   ; ignore adding the number to DH
            ADD  DH, [BX+SI]            ; DH += the odd number
SKIP:       INC  SI                     ; for pointing to the next number in the serie
            LOOP NEXT
```

```
            MOV  SUM, DH                ; store the result in the SUM variable
            MOV  AH, 4CH
            INT  21H
MAIN        ENDP
            END  MAIN
```

The aim of the program is to give an array of numbers and sum all the odd numbers together and store it in memory. So there is a loop over the numbers of the array and then for each one we should do a diction by 2 and check the remainder. If the remainder is 1, the this an odd number! There result we be in SUM variable which is word.

5.

```
;-----------------------------------------

; A PROGRAM TO COMPUTE THE FACTORIAL OF INTEGER n USING RECURSIVE PROCEDURE

; NOTE: The result must fit into a word, so for bigger results just change the type of FACT variable.

;-----------------------------------------
            .MODEL SMALL
            .STACK 64

            .DATA
N           DB   06H            ; define N
FACT        DW   ?              ; put the final result in FACT

            .CODE
MAIN        PROC FAR
            MOV  AX, @DATA
            MOV  DS, AX

            MOV  AX, 1          ; The final result will be calculated in AX
            MOV  BX, N          ; BX = N
            CALL FACTORIAL      ; Call the factorial proc and store the next instruction into the
stack
            MOV  FACT, AX

            MOV  AH, 4CH
            INT  21H
MAIN        ENDP


FACTORIAL   PROC NEAR
            CMP  BX, 1          ; if BX == 1
            JE   FINISH         ; return
            PUSH BX             ; else push the BX into the stack
            DEC  BX             ; BX -= 1
            CALL FACTORIAL      ; FACTORIAL(N-1)
            POP  BX             ; pop the top of the stack into BX
            MUL  BX             ; AX *= BX
```

```
FINISH:     RET
FACTORIAL   ENDP


            END  MAIN
```

This program will calculate the factorial of n using recursive approach. So we need to work with stack in this problem. There is a defined procedure in this program as FACTORIAL which calls itself until the BX has the value of 1. Then it returns and we pop the stack and multiply the values together and finally store the result in FACT which is defined DW. So the maximum value for FACT which can be obtained is 0FFFFH.