Lab-4

Data Transfer Instructions

Date: ...../...../.....

MOV - MOV Destination, Source.

MOV CX, 037AH → Put immediate no. 037AH to CX.

MOV BL, [437AH] → Copy byte in DS at offset 437AH to BL.

MOV AX, BX → Copy content of register BX to AX.

MOV DL, [BX] → Copy byte from memory at [BX] to DL.

MOV DS, BX → Copy word from BX to DS register.

MOV RESULT [BP], AX → Copy AX to two memory location.

MOV ES: RESULTS [BP], AX → physical address = EA + ES.

XCHG - XCHG Destination, Source.

XCHG AX, DX → Exchange word in AX with word in DX.

XCHG BL, CH → Exchange byte in BL, with byte in CH.

XCHG AL, PRICES [BX] → Exchange byte in AL with byte in memory at EA = PRICE [BX] in DS.

LEA - LEA Register, Source

LEA BX, PRICES → Load BX with offset of price in DS.

LEA BP, SS: STACK-TOP → Load BP with offset of STACK-TOP in SS.

LEA CX, [BX][DI] → Load CX with EA = [BX] + [DI]

LDS - LDS Register, Memory Add of the first word.

LDS BX, [4326] → Copy content of memory at displacement 4326H in DS to BL. content of 4327H to BH. Copy content at displacement of 4328H and 4329H in DS to DS register.

LDS SI, SPTR → Copy content of memory at displacement SPTR and SPTR+1 in DS to SI register. Copy content of memory at displacements SPTR+2 and SPTR+3 in DS to DS register. DS: SI now points at start of the desired string.

Date: ...../...../.....

ADD - ADD Destination, Source
ADC - ADC Destination, Source

ADD AL, 74H → Add immediate number 74H to content of AL. Result in AL

ADC CL, BL → Add content of BL plus carry status to content of CL.

ADD DX, BX → Add content of BX to content of DX.

ADD DX, [SI] → Add word from memory at offset [SI] in DS to content of DX.

ADD AL, PRICES [BX] → Add byte from effective address PRICES [BX] plus carry status to content of AL.

SUB - SUB Destination, Source
SBB - SBB Destination, Source

SUB CX, BX → CX - BX; Result in CX.

SBB CH, AL → Subtract content of AL and content of CF from content of CH. Result in CH.

SUB AX, 3427H → Subtract immediate number 3427H from AX.

SBB BX, [3427H] → Substract word at displacement 3427H in DS and content of CF. from BX.

SUB PRICES [BX], 04H → Substract 04 from byte at effective address PRICES [BX], if PRICES is declared with DB; substract 04 from word at effective address PRICES [BX]. if it is declared with DW.

SBB CX, TABLE [BX] → Substract word from effective address TABLE [BX] and status of CF from CX.

SBB TABLE [BX], CX → Substract CX and status of CF from word in memory at effective address TABLE [BX].

MUL - MUL Source

MUL BH → Multiply AL with BH; result in AX

MUL CX → Multiply AX with CX; Result high word in DX, low word in AX.

MUL BYTE PTR [BX] → Multiply AL with byte in DS pointed to by [BX].

MOV AX, MCAND_16 → Load 16-bit multiplicand into AX.

MOV CL, MPLIER_8 → " 8 " " " CL.

Date: ...../...../.....

DIV - DIV Source

DIV BL → Divide word in AX by byte in BL; Quotient - in AL, reminder in AH.

DIV CX → Divide down word in DX and AX by word in CX;

INC - INC Destination

INC BL → Add 1 to contains of BL register.

INC CX → Add 1 to contains of CX register.

INC BYTE PTR[BX] → Increment byte in data segment at offset contained in BX.

INC & WORD PTR[BX] → Increment the word at offset of [BX] and [BX+1] in the data segment.

INC TEMP → Increment byte or word named TEMP in the data segment. Increment byte if MAX-TEMP declared with DB. Increment word if MAX-TEMP is declared with DW.

INC PRICES [BX] → Increment element pointed to by [BX] in Array PRICES. Increment a word if prices is declared as an array of words.

DEC - DEC Destination.

DEC CL → Substract 1 from content of CL register.

DEC BP → Substract 1 from content of BP register.

DEC BYTE PTR [BX] → Substract 1 from byte at offset [BX] in DS.

DEC WORD PTR [BP] → Substract 1 from word at offset [BP] in SS.

DEC COUNT → Substract 1 from byte or word named COUNT in DS.

DAA (Decimal adjust after BCD addition).

Let AL = 59 BCD, and BL = 35 BCD

ADD AL, BL

DAA

AL = 8EH ; lower nibble > 9, add 06H to AL

AL = 94 BCD, CF = 0

Let AL = 88 BCD, and BL = 49 BCD

ADD AL, BL

DAA

AL = D1 H ; AF = 1, add 06H to AL

AL = D7 H ; upper nibble > 9, add 60H to AL

AL = 37 BCD, CF = 1

The DAA instruction updates AF, CF, SF, PF and ZF ; OF is undefined.

AAA (ASCII Adjust for Addition)

Date: ...../...../.....

Let AL = 0011 0101 (ASCII 5), and BL = 0011 1001 (ASCII 9)

ADD AL, BL

AAA

AL = 0110 1101 (6EH, which is inc BCD)

AL = 0000 0100 (unpacked BCD 4)

CF = 1 indicates answer is 14 decimal.

AND - AND Destination, Source

AND CX, [SI] → AND word in DS at offset [SI] with word in CX register, Result in CX register.

ADD BH, CL → AND byte in CL with byte in BH, Result in BH

AND BX, 00FFH → 00FFH Masks upper byte, leaves lower byte unchanged.

OR - OR destination, Source

OR AH, CL → CL ORed with AH, result in AH, CL not changed.

OR BP, SI → SI ORed with BP, result in BP, SI not changed.

OR SI, BP → BP " " SI, " " SI, BP " " .

OR BL, 80H → BL " " immediate number 80H; set MSB of ~~to~~ BL to 1.

OR CX, TABLE[SI] → CX ORed with word form effective address TABLE[SI]; content of memory is not changed.

XOR- XOR Destination, Source

XOR CL, BH → Byte in BH exclusive-ORed with byte in CL. Result in CL. BH not changed.

XOR BP, DI → Word in DI exclusive-ORed with word in BP; Result in BP. DI not changed.

XOR WORD PTR [BX], 00FFH → Exclusive-OR immediate no. 00FFH with word at offset [BX] in the data segment.

CMP - CMP destination, source

CMP AL, 01H → Compare immediate number 01H with byte in AL.

CMP BH, CL → Compare byte in CL. with byte in BH.

CMP CX, TEMP → Compare word in DS at displacement TEMP with word at CX.

TEST - Destination, Source

Date: ...../...../.....

TEST AL, BH → AND BH with AL, No result stored.
Update PF, SF, ZF.

TEST CX, 0001 H → AND CX with immediate number 0001 H; No result stored; Update PF, SF, ZF.

TEST BP, [BX][DI] → AND word wre offset [BX][DI] in DS with word in BP.

RCL - RCL Destination, Count.

RCL DX, 1 → word in DX, 1 bit left, MSB to CF, CF to LSB.

MOV CL, 4 → Load the no. of bit position to
RCL SUM [BX], CL rotate into CL.
Rotate byte or word at effective address SUM [BX] 4 bits to left.
Original bit 4 now in CF.

RCR - RCR destination, Count.

RCR BX, 1 → word in BX right 1 bit, CF to MSB, LSB to CF.

MOV CL, 4 → Load CL for rotating 4 bit position.
RCR BYTE PTR[BX], 4 → Rotate the byte at offset [BX] in DS 4 bit position right.

SAL - SAL Destination, Count

SHL - SHL " "

SAL BX, 1 → Shift word in BX. 1 bit position left, 0 in LSB.

MOV CL, 02h → Load desired no. of shift in CL

SAL BP, CL Shift word in BP left CL bit pos. 0 in LSB

SAL BYTE PTR [BX], 1 → Shift byte in DX, at offset [BX] 1 bit position left, 0 in LSB.

SAR - SAR Destination, Count

SAR DX, 1 → Shift word in DI one bit position right, new MSB = old MSB.

MOV CL, 02H → Load desired number of shifts in CL

SAR WORD PTR [BP], CL → Shift word at offset [BP] in stack segment right by two bit positions, the two MSBs are now copies of original LSB.

SHR - SHR destination, Count

SHR BP, 1 → Shift word in BP one bit position right, 0 in MSB.

MOV CL, 03H → Load desired no. of shift into CL.

SHR BYTE PTR [BX] Shift byte in DS at offset [BX] 3 bits right

Date: ...../...../.....

JBE/JNA

```
CMP AX, 4371H  →  Compare (AX - 4371H)
JBE NEXT       →  Jump to label Next if AX is
                  below or equal to 4371H
```

```
CMP AX, 4371H  →  Compare (AX - 4371H)
JNA NEXT          Jump to label NEXT if AX not
                  above 4371H
```

JG/JNLE

```
CMP BL, 39H  →  Compare by substracting 39H from BL.
JG NEXT         Jump to label NEXT if BL more positive
                than 39H
```

```
CMP BL, 39H  →  Compare by substracting 39H from BL.
JNLE NEXT       Jump to label NEXT if BL in not
                less than or equal to 39H.
```

JL/JNGE

```
CMP BL, 39H  →  Compare by substracting 39H from BL.
JL AGAIN        Jump to label AGAIN if BL more
                negative than 39H.
```

```
CMP BL, 39H  ⟶  Compare by substracting 39H from BL.
JNGE AGAIN      Jump to label AGAIN if BL not
                more positive than or equal to 39H.
```

JLE / JNG

```
CMP BL, 39H  ⟶  Compare by substracting 39H from BL.
JLE NEXT        Jump to label NEXT if BL more
                negative than or equal to 39H.
```

```
CMP BL, 39H  ⟶  Compare by substracting 39H from BL.
JNG NEXT        Jump to label NEXT if BL not more
                positive that 39H.
```

JE / ~~JNE~~ JZ

```
CMP BX, DX   ⟶  Compare (BX - DX)
JE DONE         Jump to DONE if BX - DX
```

```
IN AL, 30H   ⟶  Read data from port 8FH
SUB AL, 30H     Substract the minimum value.
JZ START        Jump to label START if the
                result of sub is 0.
```

Date: ...../...../.....

JNE/JNZ

```
IN AL, 0F8H   ⟶ Read data value from port
CMP AL, 72    Compare (AL - 72)
JNE NEXT      Jump to label NEXT if AL ≠ 72
```

```
ADD AX, 0002H ⟶ Add count factor 0002H to AX
DEC BX        Decrement BX
JNZ NEXT      Jump to label NEXT if BX ≠ 0
```

PUSH - PUSH Source.

PUSH BX → Decrement SP by 2, copy BX to stack

PUSH DS → Decrement SP by 2, copy DS to stack.

PUSH BL → Illegal: must push a word

PUSH TABLE[BX] → Decrement SP by 2, and copy word from memory in DS at EA = TABLE + [BX] to stack.

POP - POP Destination.

POP DX → Copy a word from top of stack to DX, increment SP by 2.

POP DS → Copy a word from top of stack to DS, increment by 2

POP TABLE [DX] ⟶ Copy a word from top of stack to memory in DS with EA = TABLE + [BX]; increment SP by 2.

IN - IN Accumulator, Port

IN AL, 0C8 H ⟶ Input a byte from port 0C8H to AL

IN AX, 34H ⟶ Input a word from port 34 H to AX

MOV DX, 0FF78 H ⟶ Initialize DX to point to port

IN AL, DX — Input a byte from 8 bit port 0FF78 H to AL

IN AX, DX — Input a word from 16-bit port 0FF78H to AX.

OUT = OUT Port, Accumulator

OUT 3BH, AL ⟶ Copy the content of AL to port 3BH

OUT 2CH, AX ⟶ Copy the " of AX " " 2CH.

MOV DX, 0FFF8H ⟶ Load desired port address in DX.

OUT DX, AL — Copy content of AL to port FFF8 H

OUT DX, AX — Copy content of AX to port FFF8 H.

ENDS

CODE SEGMENT ⟶ Start of logical segment containing code instruction statments

CODE ENDS — End of segment name CODE.

Date: ...../...../.....

DW.

WORDS DW 1234H, 345GH → Declare an array of 2 words and initialize them with the specified values.

STORAGE DW 100 DUP(0) → Reserve an array of 100 words of memory and initialize all 100 word with 0000. Array is named as STORAGE.

STORAGE DW 100 DUP(?) → Reserve 100 word of storage in memory and give it the name STORAGE, but leave the words un-initialized.

PROC

The PROC directive is used to identify the start of a procedure. The PROC directive follows a name you give the procedure. After the PROC directive, the term near or the term far is used to specify the name of the procedure. The ~~PRO~~ PROC directive is used with the ENDP directive to "bracket" a procedure.

ENDP

SQUARE_ROOT PROC → start of procedure

SQUARE - ROOT ENDP ⟹ end of procedure.

LABEL.

ENTRY_POINT LABEL FAR → Can jump to here from another segment.

NEXT: MOV AL, BL → Can not do a far jump directly to a label with a colon.

STACK_SEG SEGMENT STACK

DW 100 DUP(0) → Set aside 100 words for stack

STACK_TOPLABLL WORD → Give name to next location after last word in stack

STACK_SEG ENDS

INCLUDE

The directive is used to tell the assembler to insert a block of source code from the named file into the current source module.