Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Organization and Assembly Language

Assembly Assignment

Prepared by:

Student Name:	Yuna Nawahda	Student ID:	1211524
Student Name:	Nour Alaydi	Student ID:	1222810

Deadline Date:

05/02/2024

Rules and Regulations:

You are required to:

1. fill name(s), ID(s) and date in this cover page.
2. answer clearly all problems in the assignment, show your works.
3. write the name/names of the participating student(s) besides each problem. The participation percentage should be equal between you and your partner (if you have).
4. avoid cheating from other students, solution manual, and internet.
5. submit .pdf file (i.e. ID1_ID2.pdf) for the solution by accessing your RITAJ and replying on the assignment before the deadline, late submissions are not allowed.
6. do the submission by you or your partner, one submission per group is enough. If you submit twice or more, the last submission will be considered only.

Problem#1

Mention the registers related to the accumulator, index registers, program counter, and program status word? and list their specific features?

Answer#1 (*Yuna Nawahda & Nour Alaydi*):

1. **Accumulator Register (AX)**: the AX register is used primarily for arithmetic, logical, and data transfer operations. It is the default register for many operations, including multiplication and division. The AX register can be divided into two smaller registers: AH (Accumulator High) and AL (Accumulator Low) for operations that require smaller data sizes.
2. **Index Registers**:
   These registers include:
   - **Source Index (SI)**: its always pointed to the source array.
   - **Destination Index (DI)**: usually pointed to the destination array.
   - **Base Pointer (BP)**: it used access data inside the stack.
   - **Stack Pointer (SP)**: it used point the current top of stack.

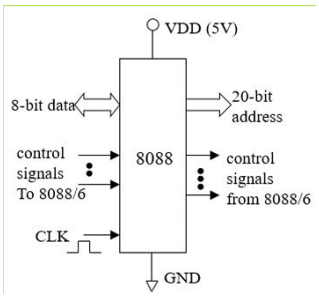
3. **Program Counter (PC) / Instruction Pointer (IP)**: it points to the address of the next instruction to be executed
4. **Program Status Word (PSW) / Flags Register**: it represented by the Flags register in x86 architecture, which includes various flags that indicate the status of the CPU and the outcome of operations.
   Key flags include:
   - **Zero Flag (ZF)**: if the result of an operation is zero.
   - **Carry Flag (CF)**: if there is a carry out or borrow into the high order bit during an arithmetic operation.
   - **Sign Flag (SF)**: see the sign of the result of an operation.
   - **Overflow Flag (OF)**: see if there was an arithmetic overflow.
   - **Interrupt Enable/Disable Flag**: it controls the handling of interrupts.

Problem#2

What is the maximum memory that can be accessed in 8086? show diagram for this.

Answer#2 (*Yuna Nawahda & Nour Alaydi*):

- The 8086 has 20 address lines, which theoretically allows it to address 2^20 = 1,048,576 bytes (1MB).



Problem#3

Find the carry flag, the auxilary carry flag, the zero flag, the overflow flag, and the sign flag after each one of the following instructions, assume AX is initialized with 1254**h** and BX is initialized with 0FFF**h**.

a. add ax, EDAB**h**

b. add ax, bx

c. add bx, F001**h**

Answer#3 (*Yuna Nawahda & Nour Alaydi):*

Initial Values:

AX = 1254h = 0001 0010 0101 0100b

BX = 0FFFh= 0000 1111 1111 1111b

Instructions:

a. **add ax, EDABh**

Operation: 1254h + EDABh

Calculation: 0001 0010 0101 0100b + 1110110110101011b= 1111111011111111b

CF	1
AF	0
ZF	0
OF	0
SF	1

b. **ADD AX, BX**

Operation: **Operation**: 1254h + BX

Calculation: 1111111011111111b + 0000 1111 1111 1111b = 0011 1111 1100 0100b

CF	0
AF	1
ZF	0
OF	0
SF	0

c. **ADD BX, F001h**

Operation: 0FFFh + F001h

Calculation: 0000111111111111 b+ 1111000000000001 b = 1111111111111111b

CF	1
AF	1
ZF	0
OF	0
SF	1

Problem#4

Based on the following listed words, specify the byte that is stored at higher memory address and the byte that is stored at lower memory address in a little endian computer. Show your results in memory diagram for each case.

a. 1234
b. ABFC
c. B100
d. B800

Answer#4 (*Yuna Nawahda & Nour Alaydi*):

- **1234**
  - Higher Memory Address: 12
  - Lower Memory Address: 34

00	**01**
34	12

- **ABFC**
  - Higher Memory Address: AB
  - Lower Memory Address: FC

02	**03**
FC	AB

- **B100**
  - Higher Memory Address: B1
  - Lower Memory Address: 00

04	**05**
00	B1

- **B800**
  - Higher Memory Address: B8
  - Lower Memory Address: 00

06	**07**
00	B8

Problem#5

Find the contents of the memory at locations 200, 201, 202, and 203. Assume the word 5678 is stored at offset 200 and the word 1234 is stored at offset 202. Show your results in memory diagram for these locations.

Answer#5 (*Yuna Nawahda & Nour Alaydi*):

big Endian:

offset	**Content (hex)**
200	0X16
201	0X2E
202	0X04
203	0XD2

little Endian:

offset	**Content (hex)**
200	0X2E
201	0X16
202	0XD2
203	0X04

Problem#6

Find the memory address for the following segment offset pairs:

a. 1DDD:0436
b. 1234:7920
c. 74F0:2123
d. 0000:6727
e. FFFF:4336
f. 1080:0100
g. AB01:FFFF

Answer#6 (*Yuna Nawahda & Nour Alaydi):*

Physical Address = (Segment Address x 16) + Offset

a. 1DDD:0436 = (1DDD * 16) + 0436 = 1E206
b. 1234:7920 = (1234 * 16) + 7920 = 019C60
c. 74F0:2123 = (74F0 * 16) + 2123 = 77023
d. 0000:6727 = (0000 * 16) + 6727 = 6727
e. FFFF:4336 = (FFFF * 16) + 4336 = 104326
f. 1080:0100 = (1080 * 16) + 0100 =10900
g. AB01:0FFF = (AB01 * 16) + 0FFF=AC00F

Problem#7

Define the "label" and how does the assembler differentiates between code labels and data labels?

Answer#7 (*Yuna Nawahda & Nour Alaydi):*

label :Act as place markers , it marks the address (offset) of code and data.
It serves as a placeholder for an address, allowing programmers to write more readable and maintainable code.

Assemblers differentiate between code labels and data labels based on context:

1. **Code Labels**: used to mark certain points in the instruction sequence, often for the purpose of jumps, loops, or branches. When the assembler encounters a label in a position that could be the target of a jump or call instruction, it treats it as a code label.
2. **Data Labels** used to identify the beginning of a data segment or a particular variable in memory. They are defined by being followed by data declaration directives (like DB, DW, DD for defining byte, word, or double-word data, respectively).

   The assembler uses the context in which the label is defined and referenced to determine its type. If a label is followed by an instruction, it's a code label; if it's followed by a data directive, it's a data label.

Problem#8

What is the effective address generated by the following instructions? Where every instruction is independent on other instructions. Assume BX=0100**h**, num1=1001**h**, [num1]=0000**h**, and SI=0100**h**.

a. mov ax, [bx+12]
b. mov ax, [bx+num1]
c. mov ax, [num1+bx]
d. mov ax, [bx+si]

Answer#8(*Yuna Nawahda & Nour Alaydi*):

BX=0100**h**, num1=1001**h**, [num1]=0000**h**, and SI=0100**h**.

a. mov ax, [bx+12]
Effective address = BX + 12h = 0100h + 0012h=0112h

b. mov ax, [bx+num1]
Effective address = BX + num1 = 0100h + 1001h =1101h

c. mov ax, [num1+bx]
Effective address = num1 + BX = 1001h + 0100h =1101h

d. mov ax, [bx+si]
Effective address = BX + SI = 0100h + 0100h = 0200h

Problem#9

Find the problems in the following instructions and fix them by changing them with one or two instruction having the same effect.

a. mov [02], [22]
b. mov [wordvar], 20
c. mov bx, al
d. mov ax, [si+di+100]

Answer#9 (*Yuna Nawahda & Nour Alaydi*):

a) cannot move data directly from one memory location to another in a single instruction.

Fix:

```
mov ax, [22]
mov [02], ax
```

b) **Fix :**

```
mov word ptr [wordvar], 20
```

c) Mismatch in operand sizes (BX is a 16-bit register, AL is an 8-bit register).

Fix :

```
xor bx, bx          Clear BX (set it to 0).
mov bl, al
```

d) do not support three-component addressing modes directly

Fix :

```
mov bx, si
add bx, di
add bx, 100
```

mov ax, [bx]

Problem#10

Which the following instruction is legal or illegal? and why?

a. inc AX
b. dec BL
c. inc [BX]
d. inc Byte PTR [BX]
e. inc DS
f. mov [BX], -1
g. mov [BX], offset I

Answer#10 (*Yuna Nawahda & Nour Alaydi*):

a. inc AX : legal
b. dec BL : legal
c. inc [BX] : legal
d. inc Byte PTR [BX] : legal
e. inc DS : illegal
f. mov [BX], -1 : legal
g. mov [BX], offset I : legal

Problem#11

Let AX=FFFF**h** and BX=0002**h**, find the result of the following instructions (**MUL** and **IMUL**), every instruction is independent on other instructions:

a. mul bl; **AX =**
b. imul bl; **AX =**
c. mul al; **AX =**
d. imul al; **AX =**
e. mul bx; **AX =** **DX =**
f. imul bx; **AX =** **DX =**

Answer#11 (*Yuna Nawahda & Nour Alaydi*):

Given:

- AX = FFFFh (11111111 11111111)
- BX = 0002h (00000000 00000010)

a. AX = FFh * 02h = 01FEh
b. AX = FFh * 02h =01FEh **(signed)**
c. AX = FFh * FFh = FE01h
d. AX = FFh * FFh = FE01h **(signed)**
e. mul bx; AX = FFFEh, DX = 0001h
f. imul bx; AX = FFFEh, DX = 0001h **(signed)**

the results didn't change

Problem#12

1. Let AX=0005**h**, BX=FFFE**h**, and DX=0000**h**, find the result of the following instructions (**DIV** and **IDIV**), every instruction is independent on other instructions:
a. div bx; **AX =** **DX=**
b. idiv bx; **AX =** **DX=**

2. Let AX=FFFB**h**, BX=0002**h**, and DX=FFFF**h**, find the result of the following instructions (**DIV** and **IDIV**), every instruction is independent on other instructions:
a. div bx; **AX =** **DX=**
b. idiv bx; **AX =** **DX=**

3. Let AX=00FB**h** and BL=FF**h**, find the result of the following instructions (**DIV** and **IDIV**), every instruction is independent on other instructions:
a. div bl; **AH =** **AL=**
b. idiv bl; **AH =** **AL=**

Answer#12(*Yuna Nawahda & Nour Alaydi*):

1 : AX=0005h, BX=FFFEh , and DX=0000h
a. div bx: AX = 0000, DX = 0005
b. idiv bx: AX = FFFF, DX = 0005

2 : Given AX=FFFBh, BX=0002h, and DX=FFFFh
a. div bx: AX = FFFE, DX = 0002
b. idiv bx: AX = FFFD, DX = 0001

3 : Given AX=00FBh and BL=FFh
a. div bl: AH = 00, AL = FF
b. idiv bl: AH = 00, AL = FF

Problem#13

1. Let AL=FF**h**, find the result of the following instructions, every instruction is independent on other instructions:

a. shr al, 1; **AL=**
b. sar al, 1; **AL=**
c. shl al, 1; **AL=**
d. sal al, 1; **AL=**

2. Let AL=0B**h** and CL=02**h**, find the result of the following instructions, every instruction is independent on other instructions:

a. shl al, 1; **AL=**
b. shl al, cl; **AL=**
c. shr al, 1; **AL=**
d. shr al, cl; **AL=**
e. rol al, 1; **AL=**
f. ror al, 1; **AL=**

Answer#13 (*Yuna Nawahda & Nour Alaydi*):

1. Given AL=FFh

a. shr al, 1 (Shift Right by 1):
- Before: 11111111
- After: 01111111 (The leftmost bit is filled with 0.)
- Result: 7Fh

b. sar al, 1 (Arithmetic Shift Right by 1):
- Before: 11111111
- After: 11111111
- Result: FFh

c. shl al, 1 (Shift Left by 1):
- Before: 11111111
- After: 11111110
- Result: FEh

d. sal al, 1 (Same as shl, Shift Left by 1):
- Result: FEh

2. Given AL=0Bh and CL=02h

a. shl al, 1 (Shift Left by 1):
- Before: 00001011
- After: 00010110
- Result: 16h

b. shl al, cl (Shift Left by CL, which is 2):
- Before: 00001011
- After: 00101100 (Two positions to the left.)
- Result: 2Ch

c. shr al, 1 (Shift Right by 1):
- Before: 00001011
- After: 00000101
- Result: 05h

d. shr al, cl (Shift Right by CL, which is 2):
- Before: 00001011
- After: 00000010 (Two positions to the right.)
- Result: 02h

e. rol al, 1 (Rotate Left by 1):
- Before: 00001011
- After rotating left: 00010110 (The leftmost bit goes to the rightmost position.)
- Result: 16h

f. ror al, 1 (Rotate Right by 1):
- Before: 00001011
- After rotating right: 10000101 (The rightmost bit goes to the leftmost position.)
- Result: 85h

☺ Good luck ☺
Ibrahim A. Nemer @ 2024