

COAL LECTURE 4

The screenshot shows a video conference interface. On the left is a code editor window titled "c01-01.asm — asm". The code is as follows:

```
ASM c01-01.asm
1 [org 0x100]
2 ; start of code
3
4 mov ax, 5          ; move the constant 5 into register ax
5 mov bx, 10         ; move the constant 10 into register bx
6 add ax, bx         ; add value of bx into the value of ax
7
8 mov bx, 15          ; move constant 15 into register bx
9 add ax, bx         ; add value of bx into the value of ax
10
11 mov ax, 0x4c00      ; exit ..
12 int 0x21           ; .. is what the OS should do for me
13
14
15
16
17
18
19
20 ; watch the listing carefully
```

The video player on the right displays a man with a beard, identified as Dr. Mohammad Nauman. The video title is "Computer Organization and Assembly Language". Below the title, it says "Dr. Mohammad Nauman FAST NUCES Peshawar Campus". At the bottom right of the video player, the URL <https://recluze.net> is visible.

- All the old lecture revision about what all the **OPP** code does and what these Instruction results in
- What we are trying to do in this lecture is that we are gonna load data from **RAM (MEMORY)** instead of hard coding it

```
ASM c01-01.asm
1 [org 0x0100]
2
3 ; start of code
4 mov ax, 87
5
6 mov ax, 5          ; move the constant 5 into ax
7 mov bx, 10
8
9 add ax, bx        ; add value of bx into the ax
10
11 mov bx, 15        ; add constant 15 into the bx
12 add ax, bx
13
14 mov ax, 0x4c00    ; exit ..
15 int 0x21          ; .. is what the OS should do
16
17
18
19
```

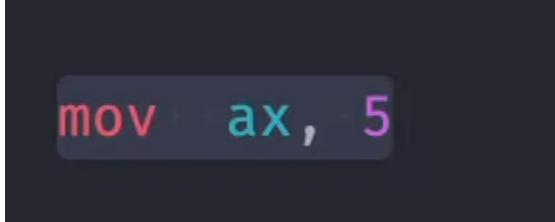
- Because of this change , by writing the (**mov ax , 87**) the address of line number **6** changes as the assembler starts assembling instruction from top to bottom
- Now instead of memorizing the address in our brain which is kinda very difficult we will give it a **LABEL** a kind of **BOOKMARK**

```
ASM c01-01.asm
1 [org 0x0100]
2
3 ; start of code
4
5 firstline:
6 mov ax, 5 ; move the constant 5 into ax
7 mov bx, 10
8
9 add ax, bx ; add value of bx into the ax
10
11 mov bx, 15 ; add constant 15 into the bx
12 add ax, bx
13
14 mov ax, 0x4c00 ; exit ..
15 int 0x21 ; .. is what the OS should do
16
17
18
19
```

- As you can see the **FIRSTLINE:** , this is not part of the code but a label that the first line starts from here

```
ASM c01-01.asm
1 [org 0x0100]
2
3 ; start of code
4
5 firstline:    mov  ax, 5          ; move the value 5 into ax
6     mov  bx, 10
7
8     add  ax, bx      ; add value of bx into the value of ax
9
10    mov   bx, 15      ; add constant 15 into the value of bx
11    add  ax, bx
12
13    mov  ax, 0x4c00    ; exit ..
14    int  0x21          ; .. is what the OS should do
15
16
17
18
19
```

- - Or you can see it this way
 - Now the address of this instruction is stored in the LABEL “**FIRSTLINE**”



```
mov ax, 5
```

- - This is nothing but a address “**B80500**”

```
4  
5 mov ax, 5 ; move the constant 5 into register ax  
6 mov bx, 10  
7  
8 add ax, bx ; add value of bx into the value of ax  
9  
10 mov bx, 15 ; add constant 15 into the value of bx  
11 add ax, bx  
12  
13 mov ax, 0x4c00 ; exit ..  
14 int 0x21 ; .. is what the OS should do for me  
15  
16  
17 num1:
```

- What are these address ?
 - These are all the addresses of RAM

dw

- DW : Define Word
 - This allocates a 16-bit memory to be used by the code when we assign something to it

```
4
5  mov  ax, 5          ; move the constant 5
6  mov  bx, 10
7
8  add  ax, bx        ; add value of bx int
9
10 mov  bx, 15         ; add constant 15 int
11 add  ax, bx
12
13 mov  ax, 0x4c00    ; exit ..
14 int  0x21           ; .. is what the OS s
15
16
17 dw   5
18
19
```

- - Now in RAM as it starts from **0x0100** the instruction on line 17 will save a 16-bit storage whose address will be **0x0113** as it is 16-bit taking 4 bytes of memory DW : defined word (16-bit) memory
 - 1 byte = 8 bits
 - 1 nibble = 4 bits
 - 2 nibbles = 1 byte
 - 0x0100 : this is two byte as **0100** are 4 nibbles and 2 bytes

```
4
5  mov  ax,  [num1]          ; move the const
6  mov  bx,  10
7
8  add  ax,  bx             ; add value of bx int
9
10 mov  bx,  15             ; add constant 15 int
11 add  ax,  bx
12
13 mov  ax,  0x4c00          ; exit ..
14 int  0x21                 ; .. is what the OS s
15
16
17 num1: dw    5
18
19
```

- - Now as discussed before remembering these addresses is a bit difficult task
 - So we save it in a **LABEL** called num1
 - As num1 acts as a pointer we write it in squared brackets [].....[num1]
 - square brackets with a label inside it means , take the value stored in the address num1 and move it into **AX**
 - Now it means take a 16 bit value from num1 and move it into AX
 - The definition of how many bits to move is defined by what register or anything in that place we are using
 - **mov ax, [num1]**
 - - This ax will define how many bits to catch and move from num1 into AX

- And this was defined by INTEL they said the value to be inserted into the register would be of the size of the register

```
[org 0x0100]

    mov  ax, [num1]          ; load fi
    mov  bx, [num2]
    add  ax, bx
    mov  bx, [num3]
    add  ax, bx
    mov  [num4], ax
    mov  ax, 0x4c00
    int  0x21

num1: dw 5
```

- - Num1 has the value 5
 - We move value 5 from num1 into the AX register
 - Thanks to little indian we will get the value 0500 in memory

```
[org 0x0100]

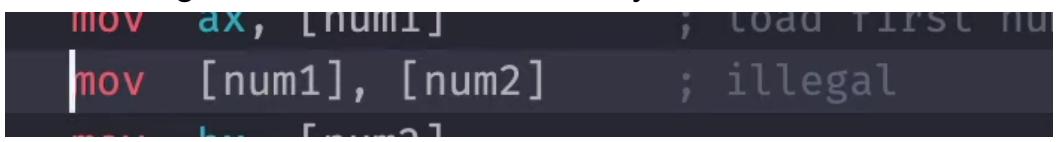
    mov  ax, [num1]          ; load first number

    mov  bx, [num2]
    add  ax, bx
    mov  bx, [num3]
    add  ax, bx
    mov  [num4], ax
    mov  ax, 0x4c00
    int  0x21

num1: dw  5
num2: dw  10
num3: dw  15
num4: dw  0
```

- - Take value from num1 address and move it into ax
 - Take value form num2 address and move it into bx
 - Take value from BX and add it into AX
 - Take value from num3 address and move it into AX
 - Now ADD AX AND BX
 - MOVE VALUE FROM AX INTO NUM4
 - Move program terminator instruction into AX

- A question might arise in mind if we can make labels then why create the AX BX register mess
 - ANSWER :
 - As we know we can either write from memory or read from memory

- We have only 1 address bus which keeps address of one thing
- So if read something from RAM and want to move it into RAM we cannot do that because we have only 1 address bus
- After reading something RAM , our CPU is not that intelligent to store the data read in some other address which is why we use registers...
- Which is why we can NEVER apply ANY instruction like MOV , ADD from MEMORY to MEMORY because we have only one ADDRESS bus and not two
- But we can move from register to register because
 - it is inside the CPU
- Whenever we are trying to move data or add as well as any other instruction that we are gonna explore you cannot have both operands as MEMORY
 - If one is memory other must be register
 - If one is register other must be memory
 - 
 - ILLEGAL

```
1 ; a program to add three numbers using memory variables
2 [org 0x0100]
3
4     mov  ax, [num1]          ; load first number in ax
5     ; mov  [num1], [num2]      ; illegal
6     mov  bx, [num2]
7     add  ax, bx
8     mov  bx, [num3]
9     add  ax, bx
10    mov  [num4], ax
11    mov  ax, 0x4c00
12    int  0x21
13
14
15 num1: dw  5
16 num2: dw  10
17 num3: dw  15
18 num4: dw  0
19
```

- - Question : As we have defined the LABEL / VARIABLE below the code how will the assembler find it out
 - Answer : Before Execution when program is loaded into MEMORY with the help of a loader all the variables if any in the program are assigned addresses of their own separately , so when we say (**mov ax, [num1]**)
 - It basically have the address , it just loads the data from that address to it

```

1 ; a program to add three numbers
2 [org 0x0100]
3
4 00000000 A1[1700]    mov ax, [num1]
5                                     ; mov [num1], [num1]
6 00000003 8B1E[1900]    mov bx, [num2]
7 00000007 01D8          add ax, bx
8 00000009 8B1E[1B00]    mov bx, [num3]
9 0000000D 01D8          add ax, bx
10 0000000F A3[1D00]     mov [num4], ax
11 00000012 B8004C     mov ax, 0x4c00
12 00000015 CD21        int 0x21
13
14
15 00000017 0500         num1: dw 5
16 00000019 0A00         num2: dw 10
17 0000001B 0F00         num3: dw 15
18 0000001D 0000         num4: dw 0
19

```

- As you can see the machine code it translates all the labels into address because it can only understand addresses
 - 4 00000000 A1[1700] mov ax, [num1]
 - 5 ; mov [num1], [num1]
- As you can see the opp code of MOVE AX has been changed
- Because now we are not taking value from some immediate operand we are taking it from address
- A1 means , it is telling the machine to take value from the address 1700
- Because of little indian it is 1700 , it in actual is 0017

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 0000	SI 0000	CS 19F5	IP 0100	Stack +0 0000	Flags 7202
BX 0000	DI 0000	DS 19F5		+2 20CD	
CX 001F	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 0 0 0

CMD >	-0005-	1 7 8 9 A B C D E
0100 A11701	MOV AX,[0117]	DS:0117 05 00 0A 00 0F 00 00 00
0103 8B1E1901	MOV BX,[0119]	DS:011F 46 F6 00 00 8B 46 F6 D1
0107 01D8	ADD AX,BX	DS:0127 E0 D1 E0 C5 5E D8 01 C3
0109 8B1E1B01	MOV BX,[011B]	DS:012F 8B 07 8B 57 02 85 D2 75
010D 01D8	ADD AX,BX	DS:0137 04 85 C0 74 1C C7 46 DC
010F A31D01	MOV [011D],AX	DS:013F 00 00 8E 5E FC 83 7D 0E
0112 B8004C	MOV AX,4C00	DS:0147 00 74 09 8B 46 F2 48 3B
0115 CD21	INT 21	DS:014F 46 F6 7E 08 B8 01 00 EB
		DS:0157 05 E9 42 01 31 C0 89 46
		DS:015F E2 8B 46 F6 D1 E0 D1 E0

2 0 1 2 3 4 5 6 7 8 9 A B C D E F	= f.Ω≡≡ i ..+...
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00ff.
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FFL.
DS:0020 FF FF FF FF FF FF FF FF FF EB 19 C0 11	6.....J.
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00
DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 up 7 dn 8 le 9 ri 10 ri

- As you can recall it was 0017 or 1700
- But as we loaded it because of ORG it changed into 1701 but because of our debugger it flipped so it becomes 0117

- After executing 0100 , we move value from 0117 to AX : 0005

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 0005	SI 0000	CS 19F5	IP 0107	Stack +0 0000	Flags 7200
BX 000A	DI 0000	DS 19F5		+2 20CD	
CX 001F	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 0 0 0
CMD >				1	7 8 9 A B C D E
0103 8B1E1901	MOV	BX,[0119]		DS:0117	05 00 0A 00 0F 00 00 00
0107 01D8	ADD	AX,BX		DS:011F	46 F6 00 00 8B 46 F6 D1
0109 8B1E1B01	MOV	BX,[011B]		DS:0127	E0 D1 E0 C5 5E D8 01 C3
010D 01D8	ADD	AX,BX		DS:012F	8B 07 8B 57 02 85 D2 75
010F A31D01	MOU	[011D],AX		DS:0137	04 85 C0 74 1C C7 46 DC
0112 B8004C	MOV	AX,4C00		DS:013F	00 00 8E 5E FC 83 7D 0E
0115 CD21	INT	21		DS:0147	00 74 09 8B 46 F2 48 3B
0117 05000A	ADD	AX,0A00		DS:014F	46 F6 7E 08 B8 01 00 EB
011A 000F	ADD	[BX],CL		DS:0157	05 E9 42 01 31 C0 89 46
				DS:015F	E2 8B 46 F6 D1 E0 D1 E0
Z	0 1 2 3 4 5 6 7	8 9 A B C D E F	= f.Ω≡■ i .+...		
DS:0000	CD 20 FF 9F 00 EA F0 FE	AD DE 1B 05 C5 06 00 00			
DS:0010	18 01 10 01 18 01 92 01	01 01 01 00 02 FF FF FFR.		
DS:0020	FF FF FF FF FF FF FF FF	FF FF FF FF EB 19 C0 11L.		
DS:0030	A2 01 14 00 18 00 F5 19	FF FF FF FF 00 00 00 00	6.....J.		
DS:0040	05 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

- It goes on , as you can see it just moves and adds as per the instruction

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 000F SI 0000 CS 19F5 IP 0109	Stack +0 0000 Flags 7204
BX 000A DI 0000 DS 19F5	+2 20CD
CX 001F BP 0000 ES 19F5 HS 19F5	+4 9FFF OF DF IF SF ZF AF PF CF
DX 0000 SP FFFE SS 19F5 FS 19F5	+6 EA00 0 0 1 0 0 0 1 0

CMD > []		000F	1	7 8 9 A B C D E
0107 01D8	ADD AX,BX	DS:0117	05 00 0A 00 0F 00 00 00	
0109 8B1E1B01	MOV BX,[011B1]	DS:011F	46 F6 00 00 8B 46 F6 D1	
010D 01D8	ADD AX,BX	DS:0127	E0 D1 E0 C5 5E D8 01 C3	
010F A31D01	MOV [011D],AX	DS:012F	8B 07 8B 57 02 85 D2 75	
0112 B8004C	MOV AX,4C00	DS:0137	04 85 C0 74 1C C7 46 DC	
0115 CD21	INT 21	DS:013F	00 00 8E 5E FC 83 7D 0E	
0117 05000A	ADD AX,0A00	DS:0147	00 74 09 BB 46 F2 48 3B	
011A 000F	ADD [BX],CL	DS:014F	46 F6 7B 08 B8 01 00 EB	
011C 0000	ADD [BX+SI],AL	DS:0157	05 E9 42 01 31 C0 89 46	
		DS:015F	E2 8B 46 F6 D1 E0 D1 E0	

2	0 1 2 3 4 5 6 7	8 9 A B C D E F	= f.Ω≡■ i .+...
DS:0000	CD 20 FF 9F 00 EA F0 FE	AD DE 1B 05 C5 06 00 00f.
DS:0010	18 01 10 01 18 01 92 01	01 01 01 00 02 FF FF FFf.
DS:0020	FF FF FF FF FF FF FF	FF FF FF FF EB 19 C0 11	δ. L.
DS:0030	A2 01 14 00 18 00 F5 19	FF FF FF FF 00 00 00 00	6.....J.
DS:0040	05 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 ... 7 up 8 dn 9 le 10 ri

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 000F	SI 0000	CS 19F5	IP 010D	Stack +0 0000	Flags 7204
BX 000F	DI 0000	DS 19F5		+2 20CD	
CX 001F	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 1

CMD >■				1	7 8 9 A B C D E
0109 8B1E1B01	MOV	BX,[011B]	DS:0117	05 00 0A 00 0F 00 00 00	
010D 01D8	ADD	AX,BX	DS:011F	46 F6 00 00 8B 46 F6 D1	
010F A31D01	MOV	[011D],AX	DS:0127	E0 D1 E0 C5 5E D8 01 C3	
0112 B8C04C	MOV	AX,4C00	DS:012F	8B 07 8B 57 02 85 D2 75	
0115 CD21	INT	21	DS:0137	04 85 C0 74 1C C7 46 DC	
0117 05000A	ADD	AX,0A00	DS:013F	00 00 8E 5E FC 83 7D 0E	
011A 000F	ADD	[BX],CL	DS:0147	00 74 09 8B 46 F2 48 3B	
011C 0000	ADD	[BX+SI],AL	DS:014F	46 F6 7E 08 B8 01 00 EB	
011E 0046F6	ADD	[BP-0A],AL	DS:0157	05 E9 42 01 31 C0 89 46	
			DS:015F	E2 8B 46 F6 D1 E0 D1 E0	

2	0 1 2 3 4 5 6 7	8 9 A B C D E F	
DS:0000	CD 20 FF 9F 00 EA F0 FE	AD DE 1B 05 C5 06 00 00	= f.Q≡■ i..+...
DS:0010	18 01 10 01 18 01 92 01	01 01 01 00 02 FF FF FFfl.
DS:0020	FF FF FF FF FF FF FF	FF FF FF FF EB 19 C0 11d. L.
DS:0030	A2 01 14 00 18 00 F5 19	FF FF FF FF 00 00 00 00	6.....J.
DS:0040	05 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 001E	SI 0000	CS 19F5	IP 0112	Stack +0 0000	Flags 7214
BX 000F	DI 0000	DS 19F5		+2 20CD	
CX 001F	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 1 1 1 0

CMD > []

010F A31D01	MOV	[011D],AX
0112 B8004C	MOV	AX,4C00
0115 CD21	INT	21
0117 05000A	ADD	AX,0A00
011A 000F	ADD	[BX],CL
011C 001E0046	ADD	[4600],BL
0120 F60000	TEST	[BX+SI],00
0123 8B46F6	MOV	AX,[BP-0A]
0126 D1E0	SHL	AX,1

1	D	E	F	0	1	2	3	4
DS:0110	1E	00	46	F6	00	00	8B	46
DS:0125	F6	D1	E0	D1	E0	C5	5E	D8
DS:012D	01	C3	8B	07	8B	57	02	85
DS:0135	D2	75	04	85	C0	74	1C	C7
DS:013D	46	DC	00	00	8E	5E	FC	83
DS:0145	7D	0E	00	74	09	8B	46	F2
DS:014D	48	3B	46	F6	7E	08	B8	01
DS:0155	00	EB	05	E9	42	01	31	C0
DS:015D	89	46	E2	8B	46	F6	D1	E0
DS:0165	D1	E0	C5	5E	D8	01	C3	8B

Z 0 1 2 3 4 5 6 7 8 9 A B C D E F = f.Ω≡■ i..+...
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00,
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF,
DS:0020 FF FF FF FF FF FF FF EB 19 C0 11 FF FF FF FF 00 00 00 00 δ. L.
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00 00 6.....J.,
DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00,

1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 up 7 dn 8 dn 9 le 10 ri

- After taking value from the AX register and moving it on the address 011D it is inverted because of little endian
 - As it is taking value from a 16-bit register which is why in memory it will occupy 16-bit memory

```
ASM c02-02.asm
1 ; a program to add three numbers accessed using a single l
2 [org 0x0100]
3
4     mov  ax, [num1]
5     mov  bx, [num1 + 2]      ; notice how we can do arithme
6     add  ax, bx              ; also, why +2 and not +1?
7     mov  bx, [num1 + 4]
8     add  ax, bx
9     mov  [num1 + 6], ax      ; store sum at num1+6
10    mov  ax, 0x4c00
11    int  0x21
12
13 num1:   dw  5
14           dw  10
15           dw  15
16           dw  0
```

- - As you can see we have written all the numbers under one label
 - And we are doing simple arithmetic , of addition in the square brackets
 - Now assmebler do not know what is num1 + 2
 - But as we know dw 5 takes 2 bytes
 - Dw 10 takes 2 bytes
 - By adding two in squared brackets it adds two bytes to the current address
 - Just like we used to do with pointers in C and C++

```
02-02.LST
1 ; a program to add three
2 [org 0x0100]
3
4 00000000 A1[1700]           mov  ax, [num1]
5 00000003 8B1E[1900]         mov  bx, [num1 + 2]
6 00000007 01D8              add   ax, bx
7 00000009 8B1E[1B00]         mov  bx, [num1 + 4]
8 0000000D 01D8              add   ax, bx
9 0000000F A3[1D00]           mov  [num1 + 6], ax
10 00000012 B8004C           mov  ax, 0x4c00
11 00000015 CD21             int   0x21
12
13 00000017 0500             num1: dw  5
14 00000019 0A00             dw    10
15 0000001B 0F00             dw    15
16 0000001D 0000             dw    0
```

○

- This is the new code whose machine code is exactly the same

```
1 ; a program to add three
2 [org 0x0100]
3
4 00000000 A1[1700]           mov  ax, [num1]
5 ; mov  [num1], [num2]
6 00000003 8B1E[1900]         mov  bx, [num2]
7 00000007 01D8              add   ax, bx
8 00000009 8B1E[1B00]         mov  bx, [num3]
9 0000000D 01D8              add   ax, bx
10 0000000F A3[1D00]          mov  [num4], ax
11 00000012 B8004C           mov  ax, 0x4c00
12 00000015 CD21             int   0x21
13
14
15 00000017 0500             num1: dw  5
16 00000019 0A00             num2: dw  10
17 0000001B 0F00             num3: dw  15
18 0000001D 0000             num4: dw  0
19
```

- This is the old file

- AS YOU CAN SEE ABOVE BOTH THE MACHINE CODE ARE EXACTLY THE SAME

```
;; a program to add three numbers accessed using a single label
[org 0x0100]

    mov  ax, [num1]
    mov  bx, [num1 + 2]
    add  ax, bx
    mov  bx, [num1 + 4]
    add  ax, bx
    mov  [num1 + 6], ax
    mov  ax, 0x4c00
    int  0x21

num1:   dw  5, 10, 15, 0
```

- - Now assembler lets us define all the words all together like C++
 - Some important CONCEPTS

```
; a program to add three numbers directly in memory
[org 0x0100]

    mov  ax, [num1]
    mov  [num1 + 6], ax      ; add this value to result

    mov  ax, [num1 + 2]
    add  [num1 + 6], ax

    mov  ax, [num1 + 4]
    add  [num1+6], ax

    mov  ax, 0x4c00
    int  0x21

num1:   dw  5, 10, 15, 0
```

mov ax, [num1]

- Moves value from num1 which is the first value 0005 into AX
- Takes value from AX and moves it into num1 + 6 address zero becomes 5

mov ax, [num1 + 2]

- Same concept takes value from num1 + 2 which is 10 into AX
- Takes value from ax and moves it into NUM 1 + 6 address , so it becomes 15

mov ax, [num1 + 4]

- It takes the value 15 and moves it into AX

add [num1+6], ax

- Simply adds value of AX into num1 + 6 address which becomes 30
 - The rest of the code is just 4C00

1	9	A	B	C	D	E	F	0
DS:0119	05	00	0A	00	0F	00	05	00
DS:0121	00	00	8B	46	F6	D1	E0	D1
DS:0129	E0	C5	5E	D8	01	C3	8B	07
DS:0131	8B	57	02	85	D2	75	04	85
DS:0139	C0	74	1C	C7	46	DC	00	00
DS:0141	8E	5E	FC	83	7D	0E	00	74
DS:0149	09	8B	46	F2	48	3B	46	F6
DS:0151	7E	08	B8	01	00	EB	05	E9
DS:0159	42	01	31	C0	89	46	E2	8B
DS:0161	46	F6	D1	E0	D1	E0	C5	5E

1	9	A	B	C	D	E	F	0
DS:0119	05	00	0A	00	0F	00	05	00

- On 9 and A we have 0005
- On B and C we have 000A
- On D and E we have 000F
- On F and 0 we have 0005

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 000A SI 0000 CS 19F5 IP 0109	Stack +0 0000 Flags 7200
BX 0000 DI 0000 DS 19F5	+2 20CD
CX 0021 BP 0000 ES 19F5 HS 19F5	+4 9FFF OF DF IF SF ZF AF PF CF
DX 0000 SP FFFE SS 19F5 FS 19F5	+6 EA00 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

CMD >	
0005	
0106 A11B01 MOV AX,[011B]	1 9 A B C D E F 0
0109 01061F01 ADD [011F],AX	DS:0119 05 00 0A 00 0F 00 05 00
010D A11D01 MOV AX,[011D]	DS:0121 00 00 8B 46 F6 D1 E0 D1
0110 01061F01 ADD [011F],AX	DS:0129 E0 C5 5E D8 01 C3 8B 07
0114 B8004C MOV AX,4C00	DS:0131 8B 57 02 85 D2 75 04 85
0117 CD21 INT 21	DS:0139 C0 74 1C C7 46 DC 00 00
0119 05000A ADD AX,0A00	DS:0141 8E 5E FC 83 7D 0E 00 74
011C 000F ADD [BX],CL	DS:0149 09 8B 46 F2 48 3B 46 F6
011E 0005 ADD [DI],AL	DS:0151 7E 08 B8 01 00 EB 05 E9
	DS:0159 42 01 31 C0 89 46 E2 8B
	DS:0161 46 F6 D1 E0 D1 E0 C5 5E

- We loaded value from 011B into AX which is 000A

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 000A	SI 0000	CS 19F5	IP 010D	Stack +0 0000	Flags 7204
BX 0000	DI 0000	DS 19F5		+2 20CD	
CX 0021	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 0 1 0

CMD > []				000F	1 9 A B C D E F 0
0109 01061F01	ADD	[011F],AX	DS:0119	05 00 0A 00 0F 00 0F 00	
010D A11D01	MOV	AX,[011D]	DS:0121	00 00 8B 46 F6 D1 E0 D1	
0110 01061F01	ADD	[011F],AX	DS:0129	E0 C5 5E D8 01 C3 8B 07	
0114 B8004C	MOV	AX,4C00	DS:0131	8B 57 02 85 D2 75 04 85	
0117 CD21	INT	21	DS:0139	C0 74 1C C7 46 DC 00 00	
0119 05000A	ADD	AX,0A00	DS:0141	8E 5E FC 83 7D 0E 00 74	
011C 000F	ADD	[BX],CL	DS:0149	09 8B 46 F2 48 3B 46 F6	
011E 000F	ADD	[BX],CL	DS:0151	7E 0B B8 01 00 EB 05 E9	
0120 0000	ADD	[BX+SI],AL	DS:0159	42 01 31 C0 89 46 E2 8B	
DS:0000	CD 20 FF 9F 00 EA F0 FE	AD DE 1B 05 C5 06 00 00	= f.Ω≡■ i ..+..		
DS:0010	18 01 10 01 18 01 92 01	01 01 01 00 02 FF FF FFft.		
DS:0020	FF FF FF FF FF FF FF	FF FF FF FF EB 19 C0 11	δ. L.		
DS:0030	A2 01 14 00 18 00 F5 19	FF FF FF FF 00 00 00 00	6.....J.		
DS:0040	05 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

1 Step	2 ProcStep	3 Retrieve	4 Help	ON	5 BRK	Menu	6	7 up	8 dn	9 le	10 ri
--------	------------	------------	--------	----	-------	------	---	------	------	------	-------

- As we add AX and 011F we get 000F which is 15
 - And the rest of the code

- SOME IMPORTANT TALKS

```
1 ; a program to add three numbers directly in memory
2 [org 0x0100]
3
4     mov  ax, [num1]
5     mov  [result], ax    ; add this value to result
6
7     mov  ax, [num1 + 2]
8     add  [result], ax
9
10    mov  ax, [num1 + 4]
11    add  [result], ax
12
13    mov  ax, 0x4c00
14    int  0x21
15
16
17 num1: dw 5, 10, 15
18 result: 0
19
```

- We should have done this for cleaner code better code
- Now variables make it easy
- In assembly language we aim to write precise and exact code as we are talking to the machine

- **NUMBER 1 :**

- READABILITY HONI CHAHİYE

- **NUMBER 2 :**

- If you are defining and using a global variable there is a 99% chance there is a flaw in your code which is why you are using global variable
- DO NOT USE GLOBAL VARIABLES

```
[org 0x0100]

    mov  ax, [num1]
    mov  [result], ax      ; add this v

    mov  ax, [num1 + 2]
    add  [result], ax

    mov  ax, [num1 + 4]
    add  [result], ax

    mov  ax, 0x4c00
    int  0x21

num1: dw 5, 10, 15
result: 0
```

-
- WHEN YOU ARE DOING ASSEMBLY LANGUAGE YOU SHOULD NOT BE DEFINING VARIABLES LIKE THIS
- This is just to make you understand

```
; a program to add three numbers using byte variables  
[org 0x0100]  
  
    mov  ax, [num1]  
  
    mov  bx, [num1+1]  
    add  ax, bx  
  
    mov  bx, [num1+2]  
    add  ax, bx  
  
    mov  [num1+3], ax  
  
    mov  ax, 0x4c00  
    int  0x21  
  
num1: db  5, 10, 15, 0  
  
; something's wrong with this code.
```

- - What will be the value of AX when line number 4 is executed ? any guesses ?
 - Well as you can see

```
num1: db  5, 10, 15, 0
```

- We have defined it as DEFINE BYTE and not DEFINE WORD
- Which is why it allocates only 1 byte which is 05 and not 0005

```
; a program to add three numbers using byte variables
[org 0x0100]

    mov  ax, [num1]

    mov  bx, [num1+1]
    add  ax, bx

    mov  bx, [num1+2]
    add  ax, bx

    mov  [num1+3], ax

    mov  ax, 0x4c00
    int  0x21

num1: db  5, 10, 15, 0

; something's wrong with this code.
```

- Now in this program if we wanna move num1 into AX , it will load 16-bit into it and not 4-bit because AX is defined as 16-bit

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 0000	SI 0000	CS 19F5	IP 0100	Stack +0 0000	Flags 7202
BX 0000	DI 0000	DS 19F5		+2 20CD	
CX 001B	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 0 0 0

CMD >	0A05	1 7 8 9 A B C D E
0100 A11701	MOV AX,[0117]	DS:0117 05 0A OF 00 89 46 E6 C7
0103 8B1E1801	MOV BX,[0118]	DS:011F 46 F6 00 00 8B 46 F6 D1
0107 01D8	ADD AX,BX	DS:0127 E0 D1 E0 C5 5E D8 01 C3
0109 8B1E1901	MOV BX,[0119]	DS:012F 8B 07 8B 57 02 85 D2 75
010D 01D8	ADD AX,BX	DS:0137 04 85 C0 74 1C C7 46 DC
010F A31A01	MOV [011A],AX	DS:013F 00 00 8E 5E FC 83 7D 0E
0112 B8004C	MOV AX,4C00	DS:0147 00 74 09 8B 46 F2 48 3B
0115 CD21	INT 21	DS:014F 46 F6 7E 00 B8 01 00 EB
		DS:0157 05 E9 42 01 31 C0 89 46
		DS:015F EZ 8B 46 F6 D1 E0 D1 E0

Z 0 1 2 3 4 5 6 7 8 9 A B C D E F	= f.Ω≡■ i .+...fl. δ. L. 6.....J.
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00	
DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF	
DS:0020 FF EB 19 C0 11	
DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00	
DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

● 1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 7 up 8 dn 9 le 10 ri

- As you can see only 1 bytes of data is distributed after the address 0117
- 05 on 0117
- 0A on 0118
- OF on 0119
- 00 on 011A

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 0A05	SI 0000	CS 19F5	IP 0103	Stack +0 0000	Flags 7200
BX 0000	DI 0000	DS 19F5		+2 20CD	
CX 001B	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 0 0 0

CMD >	OFOA	1	7 8 9 A B C D E
0100 A11701	MOV AX,[0117]	DS:0117	05 0A 0F 00 89 46 E6 C7
0103 8B1E1801	MOV BX,[0118]	DS:011F	46 F6 00 00 8B 46 F6 D1
0107 01D8	ADD AX,BX	DS:0127	E0 D1 E0 C5 5E D8 01 C3
0109 8B1E1901	MOV BX,[0119]	DS:012F	8B 07 8B 57 02 85 D2 75
010D 01D8	ADD AX,BX	DS:0137	04 85 C0 74 1C C7 46 DC
010F A31A01	MOV [011A],AX	DS:013F	00 00 8E 5E FC 83 7D 0E
0112 B8004C	MOV AX,4C00	DS:0147	00 74 09 8B 46 F2 48 3B
0115 CD21	INT 21	DS:014F	46 F6 7E 06 B8 01 00 EB
0117 050A0F	ADD AX,OFOA	DS:0157	05 E9 42 01 31 C0 89 46
		DS:015F	EZ 8B 46 F6 D1 E0 D1 E0

Z	0 1 2 3 4 5 6 7	8 9 A B C D E F	= f.Ω≡■ i .+...fl.J.
DS:0000	CD 20 FF 9F 00 EA F0 FE	AD DE 1B 05 C5 06 00 00	= f.Ω≡■ i .+...fl.J.
DS:0010	18 01 10 01 18 01 92 01	01 01 01 00 02 FF FF FF
DS:0020	FF FF FF FF FF FF FF	FF FF FF FF EB 19 C0 11	δ. L.
DS:0030	A2 01 14 00 18 00 F5 19	FF FF FF FF 00 00 00 00
DS:0040	05 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

● 1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 7 up 8 dn 9 le 10 ri

- Now as you can see it loads 0A05 into the AX register this is because AX is a 16-bit register and it will read 4 bytes of data meaning 16-bit data from memory

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD - X

AX 190F SI 0000 CS 19F5 IP 0109 Stack +0 0000 Flags 7204	BX 0FOA DI 0000 DS 19F5 +2 20CD	CX 001B BP 0000 ES 19F5 +4 9FFF OF DF IF SF ZF AF PF CF
DX 0000 SP FFFE SS 19F5 FS 19F5 +6 EA00 0 0 1 0 0 0 0 1 0		
CMD > [] 000F		
0107 01D8 ADD AX,BX	DS:0117 05 0A 0F 00 89 46 E6 C7	
0109 8B1E1901 MOV BX,[0119]	DS:011F 46 F6 00 00 8B 46 F6 D1	
010D 01D8 ADD AX,BX	DS:0127 E0 D1 E0 C5 5E D8 01 C3	
010F A31A01 MOU [011A],AX	DS:012F 8B 07 8B 57 02 85 D2 75	
0112 B8004C MOU AX,4C00	DS:0137 04 85 C0 74 1C C7 46 DC	
0115 CD21 INT 21	DS:013F 00 00 8E 5E FC 83 7D 0E	
0117 050A0F ADD AX,0FOA	DS:0147 00 74 09 8B 46 F2 48 3B	
011A 008946E6 ADD IE646+BX+DI1,CL	DS:014F 46 F6 7E 00 B8 01 00 EB	
011E C746F60000 MOV IBP-0A1,0000 DS:0157 05 E9 42 01 31 C0 89 46	DS:015F EZ 8B 46 F6 D1 E0 D1 E0	

Z 0 1 2 3 4 5 6 7 8 9 A B C D E F	= f.Ω≡■ i .+...fl.δ. L.J.
DS:0000 CD 20 FF 9F 00 EA F0 FE AD DE 1B 05 C5 06 00 00 DS:0010 18 01 10 01 18 01 92 01 01 01 01 00 02 FF FF FF DS:0020 FF FF FF FF FF FF FF FF EB 19 C0 11 DS:0030 A2 01 14 00 18 00 F5 19 FF FF FF FF 00 00 00 00 DS:0040 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 ▶ 7 up 8 dn 9 le 10 ri

- Now it will load garbage into the data why?

```

2 [org 0x0100]
3
4     mov ax, [num1]
5
6     mov bx, [num1+1]
7     add ax, bx
8
9     mov bx, [num1+2]
10    add ax, bx
11
12    mov [num1+3], ax
13
14    mov ax, 0x4c00
15    int 0x21
16
17 num1: db 5, 10, 15, 0
18
19 : something's wrong with this code.

```

- Because this should have been DW and not DB
- Which is why if you define global variable you have to keep in mind that if you make a specific size of variable you will have to remember it when moving things into it