Name: Tazmeen Afroz Roll No: 22P-9252 BAI-4A COAL-LAB-TASK-08

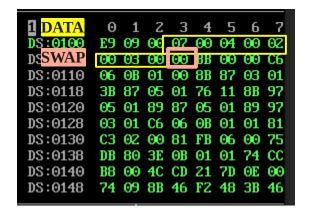
CODE:

```
[org 0x0100]
jmp start
data: dw 7, 4, 2, 3
swap: db 0 ; use this as a flag
start:
    outerloop:
       mov bx, 0
       mov byte [swap], 0
                                     ; why the "byte"?
       innerloop:
           mov ax, [data + bx]
           cmp ax, [data + bx + 2]; why did we move the value to AX?
                                     ; if we don't have to swap, we just jump over the swap thing
          jbe noswap
              ; the swap potion
              mov dx, [data + bx + 2]
              mov [data + bx + 2], ax
                                        ; again with the AX?
              mov [data + bx], dx
              mov byte [swap], 1
           noswap:
           add bx, 2
           cmp bx, 6
           jne innerloop
       ; if we didn't swap even once, we should be done
       cmp byte [swap], 1; don't need to load this in register?
       je outerloop
   ; exit system call
   mov ax, 0x4c00
    int 0x21
```

The code implements the Bubble Sort algorithm to sort an array of four word-sized (16-bit) integers (7, 4, 2, and 3) stored in memory at the data label, arranging them in ascending order (2, 3, 4, 7).

The algorithm works as follows

- 1. **Initialization**: The code starts by initializing the data to be sorted and a flag variable (swap) to track if any swaps were made during a pass through the list.
- 2. **Outer Loop**: The outer loop iterates over the list. Each iteration represents a pass through the list.
- 3. **Inner Loop**: The inner loop compares each pair of adjacent items in the list. If a pair is out of order, it swaps them.
- 4. **Swap Flag:** If any swaps are made during a pass, the swap flag is set to 1. This indicates that the list is not yet fully sorted and another pass is needed.

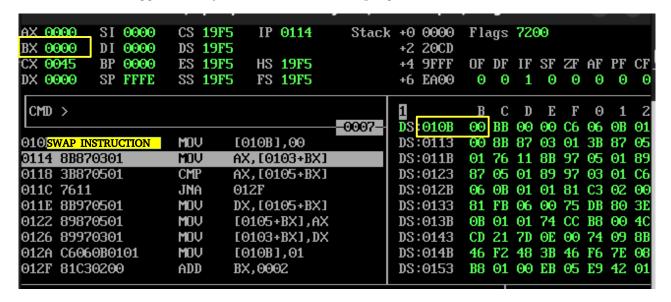


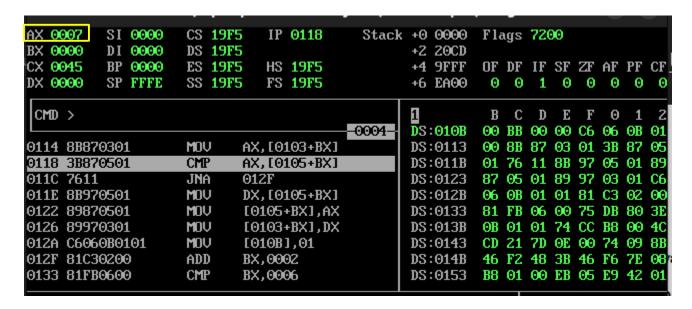
Detailed Line-by-Line Explanation

- 1. data: dw 7, 4, 2, 3: Declares a word-sized (2 bytes) array named data with the initial values 7, 4, 2, and 3. These are the numbers that will be sorted.
- 2. **swap: db 0**: Declares a byte-sized (1 byte) variable named swap and initializes it to 0. This variable is used as a flag to indicate whether any swaps were made during a pass through the list.
- 3. start::: Marks the beginning of the main part of the program.
- 4. outerloop::: Marks the beginning of the outer loop of the Bubble Sort algorithm.
- 5. mov bx, 0: Initializes the bx register to 0. The bx register is used as an index to iterate through the data array.

- 6. **mov byte [swap], 0**: Resets the swap flag to 0 at the beginning of each outer loop iteration. The byte keyword is used to specify that only the byte at the memory location pointed to by swap should be modified, not the entire word.
- 7. innerloop::: Marks the beginning of the inner loop of the Bubble Sort algorithm.
- 8. mov ax, [data + bx]: Loads the current element (the one being compared) into the ax register. The bx register is used as an index to access the elements in the array.
- 9. cmp ax, [data + bx + 2]: Compares the current element (ax) with the next element in the array ([data + bx + 2]). This is the core of the Bubble Sort algorithm, where each pair of adjacent elements is compared to determine if they are in the correct order.
- 10.**jbe noswap**: If the current element is less than or equal to the next element, this instruction jumps to the noswap label, skipping the swap operation. This is because the elements are already in the correct order, and no swap is needed.
- 11.mov dx, [data + bx + 2]: If a swap is needed, this instruction moves the next element into the dx register for temporary storage. This is done to preserve the value of the next element before it is overwritten by the current element.
- 12.mov [data + bx + 2], ax: This instruction swaps the current element with the next element by moving the value in ax (the current element) to the position of the next element.
- 13.mov [data + bx], dx: This instruction completes the swap by moving the original next element (which was temporarily stored in dx) to the position of the current element.
- 14.mov byte [swap], 1: This instruction sets the swap flag to 1, indicating that a swap was made during this pass through the list. This is crucial for determining whether another pass through the list is needed.
- 15.**noswap::**: Marks the point in the code where the swap operation is skipped if no swap is needed.
- 16.add bx, 2: Moves to the next pair of elements in the array. The bx register is incremented by 2 after each comparison and swap operation to move to the next pair of elements.
- 17.cmp bx, 6: Checks if the end of the array has been reached. Since the data array has 4 elements and each element is 2 bytes, the loop should iterate 3 times (0, 2, 4), so the comparison is with 6 to ensure the loop ends correctly.
- 18.jne innerloop: If not at the end, jumps back to the innerloop label, continuing the inner loop.
- 19.cmp byte [swap], 1: Checks if any swaps were made during the last pass. If a swap was made, the flag will be 1, and the outer loop should continue.

- 20.**je outerloop**: If the swap flag is 1, this instruction jumps back to the outerloop label, starting another pass through the list.
- 21.mov ax, 0x4c00: Prepares the system call to exit the program.
- 22.int 0x21: Triggers the system call to exit the program.





As 07 is not below and equal than 4 so 4 and 7 positions will be swapped.

AX 0007 SI 0000	CS 19F5	IP 011C	Stack	+0 0000	Flags	7204		
BX 0000 DI 0000	DS 19F5			+2 20CD				
CX 0045 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 >				1	0 1	2 3	4 5	6 7:
				DS:0100				00 02
0118 3B870501	CMP i	AX,[0105+BX]		DS:0108		1.0		00 C6-
011C 7611		912F		DS:0110	06 OB	01 00	8B 87	03 01
011E 8B970501	MOV	DX,[0105+BX]	— I	DS:0118	3B 87	05 01	76 11	8B 97
0122 89870501	MOV	[0105+BX],AX		DS:0120	05 01	89 87	05 01	89 97
0126 89970301	MOV	[0103+BX],DX		DS:0128	03 01	C6 06	OB 01	01 81
012A C6060B0101	MOV	[010B],01		DS:0130	C3 02	00 81	FB 06	00 75
012F 81C30200	ADD 1	BX,0002		DS:0138	DB 80	3E 0B	01 01	74 CC
0133 81FB0600	CMP I	BX,0006		DS:0140	B8 00	4C CD	21 7D	0E 00
0137 75DB	JNZ	0114		DS:0148	74 09	8B 46	F2 48	3B 46-

After swapping

						_						
AX 0007 SI 0000	CS 19F5		Stack			lags	72	94				
BX 0000 DI 0000	DS 19F5			+2 2		e ne	113	O.E.	-ZE	ΔE	DE	OF
CX 0045 BP 0000	ES 19F5				OFFF O			SF 0	ΖF	AF O	PF	Cr.
DX 0004 SP FFFE	SS 19F5	FS 19F5		+6 E	инов	0 0	<u> </u>	U	0	U	1	0
CMD >				1		0 1	2	3	4	5	6	7
L				DS:0	0100 E	9 09	90	04	00	07	00	02
012A C6060B0101	MOV	[010B],01		DS:0	0108 0	0 03	00	01	BB	00	00	C6
012F 81C30200	ADD	BX,0002		DS:0	0110 0	6 OE	01	90	8B	87	03	01
0133 81FB0600	CMP	BX,0006		DS:0)118 3	B 87	05	01	76	11	8B	97
0137 75DB	JNZ	0114		DS:0	0120 0	5 01	89	87	05	01	89	97
0139 803E0B0101	CMP	[010B],01		DS:0	0128 0	3 01	. C6	96	ΘB	01	01	81
013E 74CC	JZ	010C		DS:0	0130 C	3 02	00	81	\mathbf{FB}	96	$\Theta\Theta$	75
0140 B8004C	MOV	AX,4C00		DS:0)138 D	B 80	3E	ΘB	01	01	74	CC
0143 CD21	INT	21		DS:0	0140 B	B 00	4C	CD	21	7D	ΘE	00
0145 7D0E	JNL	0155		DS:0	148 7	4 09	8B	46	FZ	48	3B	46-

Next iteration

						•					_		
AX 0007 SI 0000	CS 19F5	IP 0133	Stack	+0	0000	Flā	ags	720	90				-
BX 0002 DI 0000	DS 19F5			+2	20CD								
CX 0045 BP 0000	ES 19F5	HS 19F5		+4	9FFF	\mathbf{OF}	DF	\mathbf{IF}	SF	ΖF	ΑF	\mathbf{PF}	CF.
DX 0004 SP FFFE	SS 19F5	FS 19F5		+6	EA00	0	0	1	0	0	0	0	Θ
CMD >■				1		0	1	z	3	4	5	6	 75
					0100	E9	09	00			07	00	023
012F 81C30200	ADD BX,	0002		DS	0108	00	03	00	01	BB	00	00	C63
0133 81FB0600	CMP BX,	0006		DS	:0110	06	ΘB	01	00	8B	87	03	015
0137 75DB	JNZ 011	4		DS	:0118	3B	87	05	01	76	11	8B	97
0139 803E0B0101	CMP [01	OB],O1		DS	:0120	05	01	89	87	05	01	89	975
013E 74CC	JZ 010	С		DS	:0128	03	01	C6	96	ΘB	01	01	815
0140 B8004C	MOV AX,	4000		DS	:0130	С3	02	$\Theta\Theta$	81	$\mathbf{F}\mathbf{B}$	06	$\Theta\Theta$	758
0143 CD21	INT 21			DS	:0138	DB	80	ЗE	ΘB	01	01	74	CCE
0145 7D0E	JNL 015	5		DS	:0140	B8	00	4 C	CD	21	7D	ΘE	003
0147 007409	ADD [SI	+091,DH		DS	:0148	74	09	8B	46	FZ	48	3B	46

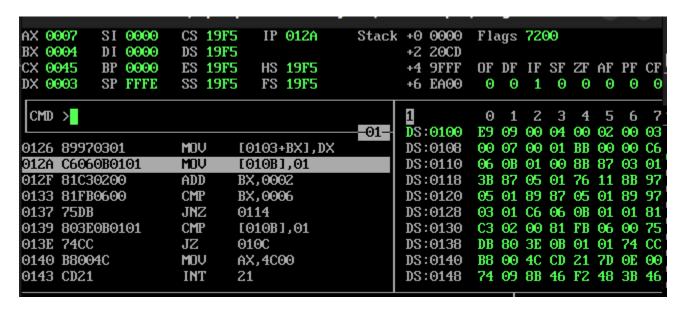
swapping

		•		•				_	
AX 0007 SI 0000	CS 19F5	IP 012A	Stack	+0 0000	Flags	7204			
BX 0002 DI 0000	DS 19F5			+2 20CD					
CX 0045 BP 0000	ES 19F5	HS 19F5		+4 9FFF	OF DF	IF SF	ZF 1	AF P	F CF
DX 0002 SP FFFE	SS 19F5	FS 19F5		+6 EA00	0 0	1 0	0	0	1 0
CMD >				1	0 1	2 3	4	5 1	5 7
			01-	DS:0100	E9 09	00 04	00	0Z 0	9 07
0126 89970301	MOV	[0103+BX],DX		DS:0108	00 03				
012A C6060B0101	MOV	[010B],01		DS:0110	06 OB	01 00	8B (B7 00	3 01
012F 81C30200	ADD	BX,0002		DS:0118	3B 87	05 01	76	11 8	8 97
0133 81FB0600	CMP	BX,0006		DS:0120	05 01	89 87	05 (01 8	97
0137 75DB	JNZ	0114		DS:0128	03 01	C6 06	ΘB (01 0	1 81
0139 803E0B0101	CMP	[010B],01		DS:0130	C3 02	00 81	FB (06 O	9 75
013E 74CC	JZ	010C		DS:0138	DB 80	3E 0B	01	01 7 ₀	4 CC
0140 B8004C	MOV	AX,4C00		DS:0140	B8 00	4C CD	21	7D 0	E 00
0143 CD21	INT	21		DS:0148	74 09	8B 46	FZ ·	48 3	8 46

3rd Iteration

		•	•		•								
AX 0007 SI 0000	CS 19F	5 IP 0133	Stack	+0	0000	Flā	ags	720	90				
BX 0004 DI 0000	DS 19F	5		+2	20CD								
CX 0045 BP 0000	ES 19F	5 HS 19F5		+4	9FFF	\mathbf{OF}	DF	\mathbf{IF}	SF	ΖF	ΑF	PF	CF.
DX 0002 SP FFFE	SS 19F	5 FS 19F5		+6	EA00	0	0	1	0	0	0	0	0
CMD >				1		0	1	2.	3	4	5	6	7
0.12					0100	_	09	_	_	-	02	_	07
012F 81C30200	ADD	BX,000Z			:0108						00		_
0133 81FB0600	CMP	BX,0006		DS:	:0110	06	ΘB	01	00	8B	87	03	01
0137 75DB	JNZ	0114		DS:	:0118	3B	87	05	01	76	11	8B	97
0139 803E0B0101	CMP	[010B],01		DS:	:0120	05	01	89	87	05	01	89	97
013E 74CC	JZ	010C		DS:	:0128	03	01	C6	06	ΘB	01	01	81
0140 B8004C	MOV	AX,4C00		DS:	:0130	C3	02	00	81	\mathbf{FB}	06	00	75
0143 CD21	INT	21		DS:	:0138	DB	80	3E	ΘB	01	01	74	CC
0145 7D0E	JNL	0155		DS:	:0140	B8	00	4 C	CD	21	7D	ΘE	00
0147 007409	ADD	[SI+091,DH		DS:	:0148	74	09	8B	46	FZ	48	3B	46-

Swapping



						•					_		
AX 0007 SI 0000	CS 19F5	IP 0133	Stack	(+0	0000	Fla	ags	720	94				
BX 0006 DI 0000	DS 19F5	5		+2	20CD								
CX 0045 BP 0000	ES 19F5	HS 19F5		+4	9FFF	\mathbf{OF}	DF	\mathbf{IF}	SF	ΖF	ΑF	PF	CF
DX 0003 SP FFFE	SS 19F5	FS 19F5		+6	EA00	0	Θ	1	Θ	Θ	0	1	Θ
CMD >				1		0	1	2	3	4	5	6	7
					0100	E9	09	$\overline{00}$	04	00	02	00	03
012F 81C30Z00	ADD	BX,0002			0108	00							C6-
0133 81FB0600	CMP	BX,0006		DS:	0110	06	ΘB	01	00	8B	87	03	01
0137 75DB	JNZ	0114		DS:	0118	3B	87	05	01	76	11	8B	97
0139 803E0B0101	CMP	[010B],01		DS:	0120	05	01	89	87	05	01	89	97
013E 74CC	JZ	010C		DS:	0128	03	01	C6	06	ΘB	01	01	81
0140 B8004C	MOV	AX,4C00		DS:	0130	С3	02	00	81	$\mathbf{F}\mathbf{B}$	06	$\Theta\Theta$	75
0143 CD21	INT	21		DS:	0138	DB	80	3E	ΘB	01	01	74	cc_{ℓ}
0145 7D0E	JNL	0155		DS:	0140	B8	00	4 C	CD	21	7D	ΘE	00
0147 007409	ADD	[SI+091,DH		DS:	0148	74	09	8B	46	FZ	48	3B	46
							_						Ł

															_		
AX 0000	SI	9000	CS	19F5	ΙP	0100	Stack	+0	0000	Flā	ags	720	92				
BX 0000	DI (9000	DS	19F5				+2	20CD		_						-
CX 0000	BP (9000	ES	19F5	HS	19F5		+4	9FFF	OF	DF	\mathbf{IF}	SF	ZF	ΑF	\mathbf{PF}	CF
DX 0000	SP I	FFFE	SS	19F5	FS	19F5		+6	EA00	0	Θ	1	Θ	Θ	Θ	0	Θ
CMD >							$\neg \neg$	1		0	1	2	3	4	5	6	7
0.2									0100	E9	09			00	03	00	04
P	rogram	termi	inate	d OK					:0108		07						
0100 E9			JMP		010C			DS	:0110		ΘB						
0103 020	00		ADD) f	L, [B	X+SII		DS	:0118	3B	87	05	01	76	11	8B	97
0105 030	00		ADD) f	X, EB	X+S11		DS	:0120	05	01	89	87	05	01	89	97
0107 040	00		ADD) f	L,00			DS	:0128	03	01	C6	06	ΘB	01	01	81
0109 07			POF	·	S			DS	:0130	С3	02	99	81	$\mathbf{F}\mathbf{B}$	06	99	75
010A 00	90		ADD) [BX+S	I],AL		DS	:0138	DB	80	ЗЕ	ΘВ	01	01	74	CC.
010C BB	0000		MOL		X,00			DS	:0140	B8	00	4 C	CD	21	7D	ΘE	00
010F C6	060B010	90	MOL		010B			DS	:0148	74	09	8B	46	FZ	48	3B	46
											_						