

// BINARY SEARCH ALGORITHM

```
LXI SP,4000    // Initializing Stack Pointer

MVI C,07       // No of elements

MVI B,07       // Key element

LXI H,2001     // Initializing Register pair to 2001H


UPLOOP:        PUSH H

MOV A,L        // Moving content of L to A

ADD C// Adding with C and the result is stored in A

RAR    // Dividing A by 2

MOV L,A        // Moving content of A to L

MOV A,B        // Moving content of B to A for comparison

CMP M          // Comparing

JZ FOUND       // IF Equal --> Jump to Found

JNC RIGHTLOOP  // If less than the key element --> Jump to RIGHT

JC LEFTLOOP    // If more than the key element --> Jump to LEFT


LEFTLOOP:      MOV A,L    // Moving content of L to A

CMP C          // Comparing with C to check if the test case has been exhausted or not

JZ NOT_FOUND   // If exhausted or if found equal --> Jump to NOTFOUND

DCX H// Decrementing HL by 1

MOV C,L        // Moving content of L to C

POP H// Moving the value stored in stack to HL pair

JMP UPLOOP     // Jumping unconditionally to LOOP


RIGHTLOOP:     MOV A,L    // Moving content of L to A

CMP C          // Comparing with C to check if the test case has been exhausted or not

JZ NOT_FOUND   // If exhausted or if found equal --> Jump to NOTFOUND

INX H // Incrementing HL register pair to 1

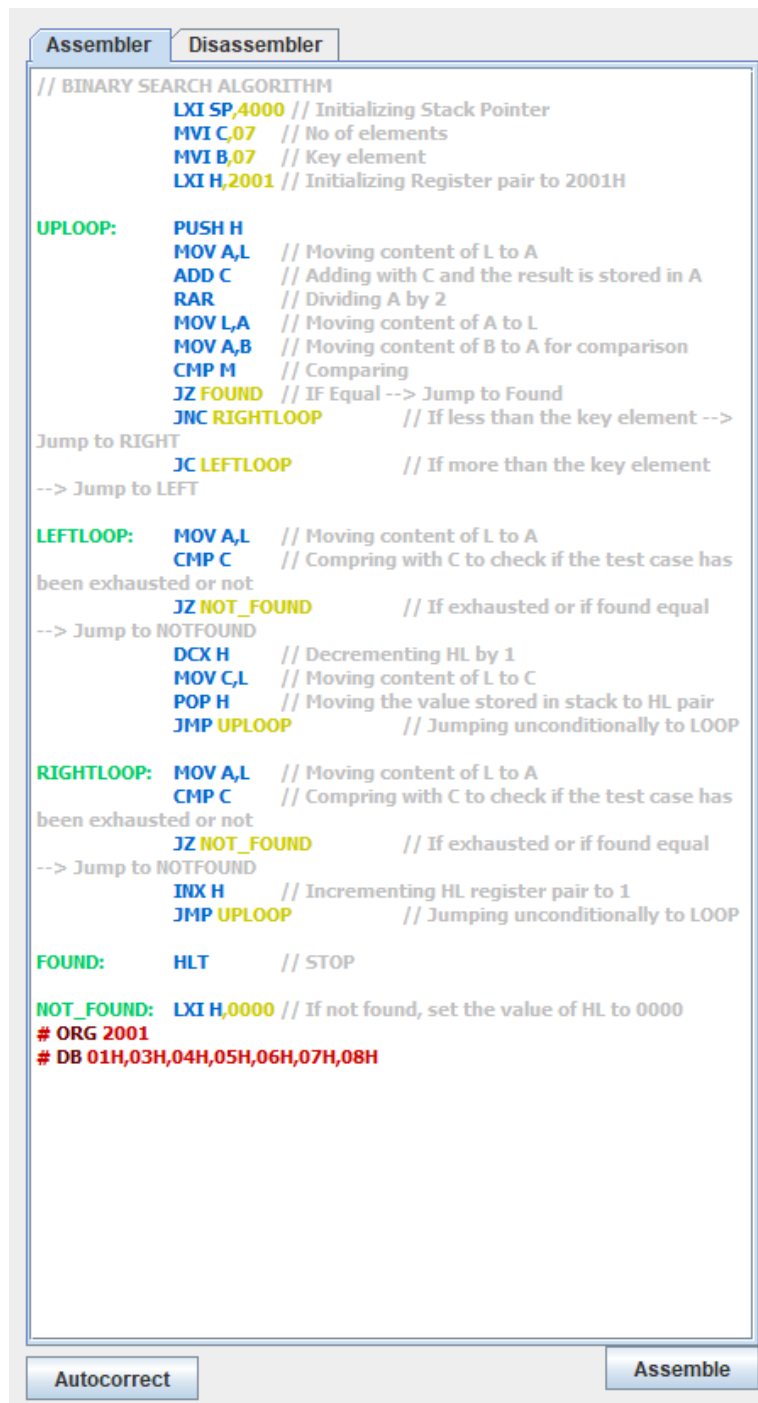
JMP UPLOOP     // Jumping unconditionally to LOOP
```

FOUND: HLT // STOP

NOT_FOUND: LXI H,0000 // If not found, set the value of HL to 0000

ORG 2001

DB 01H,03H,04H,05H,06H,07H,08H



The screenshot shows a software window with two tabs: 'Assembler' and 'Disassembler'. The 'Assembler' tab is active, displaying assembly code for a binary search algorithm. The code is color-coded: comments are in black, labels and jump targets are in green, and instructions are in blue. The algorithm includes initialization, a main loop (UPLOOP), and two sub-loops (LEFTLOOP and RIGHTLOOP) for searching the array. It concludes with a 'FOUND' label and a 'NOT_FOUND' label. At the bottom of the window, there are two buttons: 'Autocorrect' and 'Assemble'.

```
// BINARY SEARCH ALGORITHM
LXI SP,4000 // Initializing Stack Pointer
MVI C,07    // No of elements
MVI B,07    // Key element
LXI H,2001  // Initializing Register pair to 2001H

UPLOOP:     PUSH H
            MOV A,L    // Moving content of L to A
            ADD C      // Adding with C and the result is stored in A
            RAR        // Dividing A by 2
            MOV L,A    // Moving content of A to L
            MOV A,B    // Moving content of B to A for comparison
            CMP M      // Comparing
            JZ FOUND   // IF Equal --> Jump to Found
            JNC RIGHTLOOP // If less than the key element -->

            // Jump to RIGHT
            JC LEFTLOOP // If more than the key element
            --> Jump to LEFT

LEFTLOOP:   MOV A,L    // Moving content of L to A
            CMP C      // Comparing with C to check if the test case has
            // been exhausted or not
            JZ NOT_FOUND // If exhausted or if found equal
            --> Jump to NOTFOUND
            DCX H      // Decrementing HL by 1
            MOV C,L    // Moving content of L to C
            POP H      // Moving the value stored in stack to HL pair
            JMP UPLOOP  // Jumping unconditionally to LOOP

RIGHTLOOP:  MOV A,L    // Moving content of L to A
            CMP C      // Comparing with C to check if the test case has
            // been exhausted or not
            JZ NOT_FOUND // If exhausted or if found equal
            --> Jump to NOTFOUND
            INX H      // Incrementing HL register pair to 1
            JMP UPLOOP  // Jumping unconditionally to LOOP

FOUND:      HLT        // STOP

NOT_FOUND:  LXI H,0000 // If not found, set the value of HL to 0000
# ORG 2001
# DB 01H,03H,04H,05H,06H,07H,08H
```

Autocorrect Assemble

Memory Address	Value
0018	1A
001A	7D
001B	B9
001C	CA
001D	2F
001F	2B
0020	4D
0021	E1
0022	C3
0023	0A
0025	7D
0026	B9
0027	CA
0028	2F
002A	23
002B	C3
002C	0A
002E	76
002F	21
2001	01
2002	03
2003	04
2004	05
2005	06
2006	07
2007	08

Numbers are stored from 2001 to 2007

Register	Value	7	6	5	4	3	2	1	0
Accumulator	04	0	0	0	0	0	1	0	0
Register B	07	0	0	0	0	0	1	1	1
Register C	07	0	0	0	0	0	1	1	1
Register D	00	0	0	0	0	0	0	0	0
Register E	00	0	0	0	0	0	0	0	0
Register H	20	0	0	1	0	0	0	0	0
Register L	01	0	0	0	0	0	0	0	1
Memory(M)	01	0	0	0	0	0	0	0	1

Initially the value stored in HL pair is 2001

Register	Value	7	6	5	4	3	2	1	0
Accumulator	04	0	0	0	0	0	1	0	0
Register B	07	0	0	0	0	0	1	1	1
Register C	07	0	0	0	0	0	1	1	1
Register D	00	0	0	0	0	0	0	0	0
Register E	00	0	0	0	0	0	0	0	0
Register H	20	0	0	1	0	0	0	0	0
Register L	04	0	0	0	0	0	1	0	0
Memory(M)	05	0	0	0	0	0	1	0	1

Now HL has the value 2004 (which is $(A+C)/2$)

Register	Value	7	6	5	4	3	2	1	0
Accumulator	04	0	0	0	0	0	1	0	0
Register B	07	0	0	0	0	0	1	1	1
Register C	07	0	0	0	0	0	1	1	1
Register D	00	0	0	0	0	0	0	0	0
Register E	00	0	0	0	0	0	0	0	0
Register H	20	0	0	1	0	0	0	0	0
Register L	05	0	0	0	0	0	1	0	1
Memory(M)	06	0	0	0	0	0	1	1	0

Now HL has the value 2006

Register	Value	7	6	5	4	3	2	1	0
Accumulator	07	0	0	0	0	0	1	1	1
Register B	07	0	0	0	0	0	1	1	1
Register C	07	0	0	0	0	0	1	1	1
Register D	00	0	0	0	0	0	0	0	0
Register E	00	0	0	0	0	0	0	0	0
Register H	20	0	0	1	0	0	0	0	0
Register L	06	0	0	0	0	0	1	1	0
Memory(M)	07	0	0	0	0	0	1	1	1

Finally the number has been found and HL has the value 2006

Registers :									
Register	Value	7	6	5	4	3	2	1	0
Accumulator	07	0	0	0	0	0	1	1	1
Register B	07	0	0	0	0	0	1	1	1
Register C	07	0	0	0	0	0	1	1	1
Register D	00	0	0	0	0	0	0	0	0
Register E	00	0	0	0	0	0	0	0	0
Register H	20	0	0	1	0	0	0	0	0
Register L	06	0	0	0	0	0	1	1	0
Memory(M)	07	0	0	0	0	0	1	1	1

Resister	Value	S	Z	*	AC	*	P	*	CY
Flag Resister	54	0	1	0	1	0	1	0	0

Type	Value
Stack Pointer(SP)	3FFC
Memory Pointer (HL)	2006
Program Status Word(PSW)	0754
Program Counter(PC)	002E
Clock Cycle Counter	175
Instruction Counter	27

SOD	SID	INTR	TRAP	R7.5	R6.5	R5.5
0	0	0	0	0	0	0

For SIM instruction

SOD	SDE	*	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

For RIM instruction

SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

No. Converter Tool :

Hexadecimal	Decimal	Binary
0	0	0

Final values in all the register pairs