

Project Report

Project Title: Implementation of Binary Search Assembly Code 8086

Course Code: CSE360

Course Title: Computer Architecture

Section: 1

Semester: Summer 2023

Group no: 08

Group members Information

Student's Name	Student's Id
Sadia Afrin Raisa	2019-1-60-146
Md. Tohidul Haque Sagar	2019-1-60-156
Bahauddin Ahmed	2020-1-60-271

Submitted To:

Dr. Md. Nawab Yousuf Ali
Professor, Department of computer science and engineering
East West University

Submission : 10 September, 2023

Project Title: Implementation of Binary Search Assembly Code 8086

Objective:

The objective of implementing a binary search algorithm in assembly language for the Intel 8086 processor (or any other processor) is to efficiently search for a specific element in a sorted array or list. Binary search is a divide-and-conquer algorithm that can significantly reduce the number of comparisons needed to find an element compared to linear search. Binary search is a highly efficient searching algorithm, particularly for large datasets. It reduces the number of comparisons required to find an element from $O(n)$ in a linear search to $O(\log n)$ in a binary search. Binary search provides consistent and deterministic performance, as the number of comparisons required is logarithmic, making it suitable for real-time systems and applications with strict timing requirements. Overall, the objective of implementing binary search in the 8086 assembly language is to take advantage of its efficiency, memory optimization, and deterministic performance characteristics to efficiently locate elements in a sorted array or list within the constraints of the hardware and software environment.

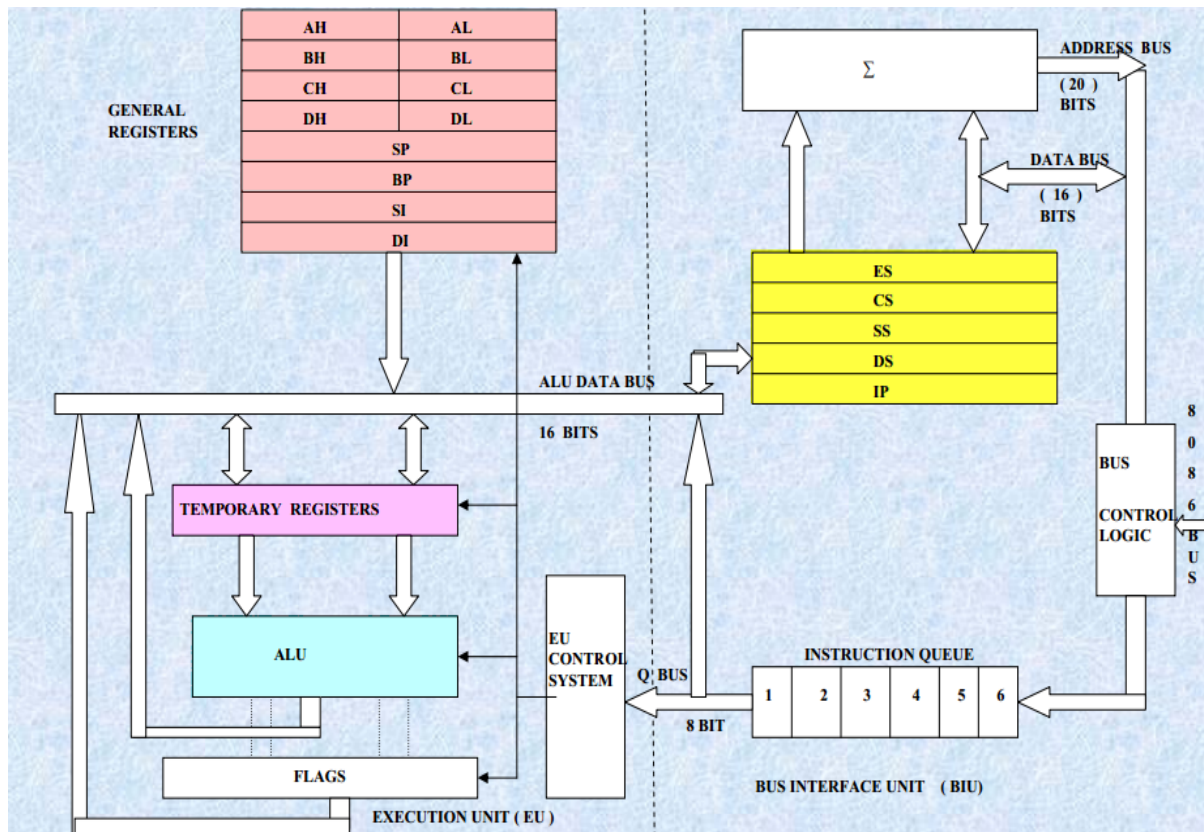
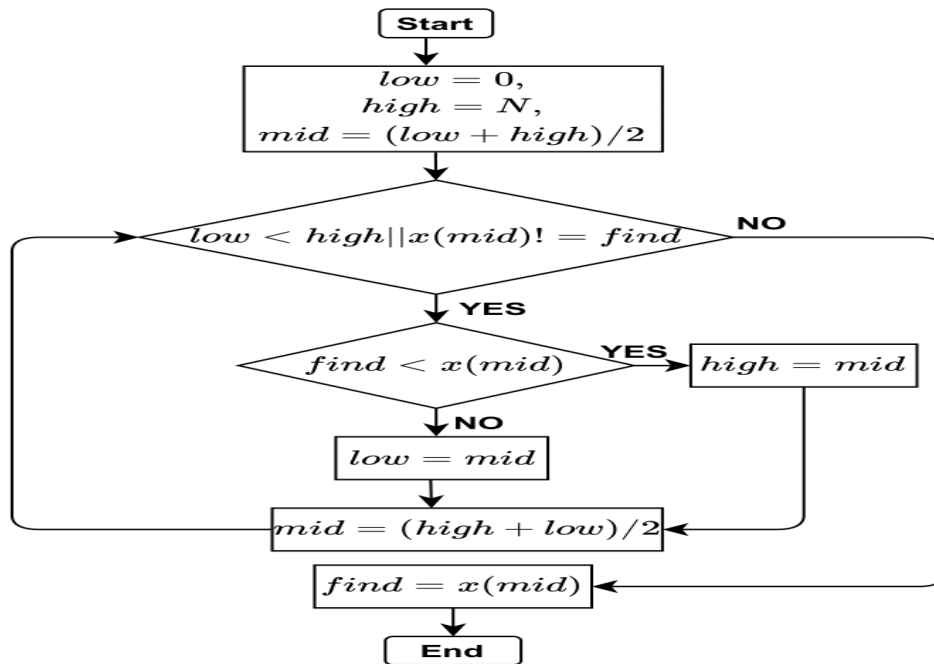
Theory:

Binary search is a divide-and-conquer technique that divides the search interval in half periodically on sorted arrays. It compares the interval's middle element to the target element and narrows the search until the target element is discovered or the search interval is empty. The binary search technique for the 8086 microprocessor is implemented in this assembly code. It closely follows the pseudocode, making comparisons and updating as needed until it discovers the target element or determines that it is not in the array. If the target is located, the "found" flag is set to 1, and the target index (in BX) can be utilized for further processing.

Design :

Flowchart & Architecture:

The flowchart of our project is given below:



Implementation:

Implementing binary search in assembly language for the Intel 8086 processor involves writing code that performs the binary search algorithm on a sorted array of data. This assembly code performs a binary search in the sorted array. You can replace the target value and the array with your specific data. The code initializes the lower bound and upper bound and repeatedly calculates the middle index to compare the middle element with the target value. Depending on the comparison result, it updates the bounds and continues the search until the element is found or the bounds overlap (not found).

Code :

```
; Initializing necessary data
.data
Out1 db "Element Found At Position : "
Out2 db " !!$"
failOut db "Searched Element Not Found!!!"
ArrList dw 12h, 28h, 100h, 212h, 228h, 534h, 550h, 888h, 980h
ArrSz dw ($-ArrList)/2
Key equ 228h

; Starting main code
.code
main proc
mov ax, @data
mov ds, ax

    mov bx, 00      ; Lower bound to base reg
    mov dx, ArrSz   ; Upper bound to data reg
    mov cx, Key     ; Key to count reg

; Main function
mainFunc:
    cmp bx, dx
    ja notMatched
    mov ax, bx
    add ax, dx
    shr ax, 1
    mov si, ax
    add si, si
    cmp cx, ArrList[si]
    jae greaterEq
    dec ax
    mov dx, ax
```

```

        jmp mainFunc
; If arrList[mid] <= key
greaterEq:
        je ifMatched
        inc ax
        mov bx, ax
        jmp mainFunc
; If arrList[mid] = key
ifMatched:
        add al, 01
        add al, 48
        lea si, Out2
        mov [si], al
        lea dx, Out1
        jmp dispOut
notMatched:
        lea dx, failOut
dispOut:
        mov ah, 09h
        int 21h
        mov ah, 4ch
        int 21h
        main endp
end main

```

Description:

- The main procedure (main proc) begins by setting up the data segment register (ds) with the address of the data section.
- `mov bx, 00` initializes bx as the lower bound (0) for the binary search.
- `mov dx, ArrSz` initializes dx with the upper bound, which is the size of the array.
- `mov cx, Key` initializes cx with the key to be searched.
- `cmp bx, dx` compares the lower bound (bx) with the upper bound (dx). If bx is greater than dx, it means the search has exhausted the possibilities, so it jumps to `notMatched`.
- `mov ax, bx` copies the lower bound to ax.
- `add ax, dx` adds the upper bound to ax.
- `shr ax, 1` divides ax by 2 by performing a right shift.
- `mov si, ax` stores the calculated middle index in si.
- `add si, si` is used to convert the middle index to an array index (since each element in the array is 2 bytes).
- `cmp cx, ArrList[si]` compares the key (cx) with the value at the middle index in the array. If the key is greater than or equal to the middle element, it jumps to `greaterEq`.
- If the key is less than the middle element, it decrements the upper bound (dx) and continues the binary search by jumping back to `mainFunc`.

- If the key is equal to the middle element, it jumps to ifMatched.
- If the key is greater than the middle element, it increments the lower bound (bx) and continues the binary search by jumping back to mainFunc.
- If the key is found, it increments the character in al to convert it to a digit, appends it to the end of the string in Out2, and loads the address of Out1 into dx. Then, it jumps to dispOut.
- If the key is not found (notMatched), it loads the address of failOut into dx.
- dispOut is used to display the appropriate message. It uses DOS interrupt 21h to print the message.
- main endp marks the end of the main procedure.
- end main marks the end of the program.

Experimental Result:

```

edit: E:\14th semester\cse 360\project360.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  options  help  about
001 ;initializing necessary data
002 .data
003 Out1 db "Element Found At Position : "
004 Out2 db " !!"
005 failOut db "Searched Element Not Found!!!"
006 ArrList dw 12h,28h,100h,212h,228h,534h,550h,888h,980h
007 ArrSz dw ($-ArrList)/2
008 Key equ 534h
009
010 ;starting main code
011 .code
012 main proc
013     mov ax, @data
014     mov ds, ax
015
016     mov bx, 00h ;Lower bound to base reg
017     mov dx, ArrSz ;Upper bound to data reg
018     mov cx, Key ;Key to count reg
019 ;main function
020 mainFunc:
021     cmp bx, dx
022     ja notMatched
023
024     mov ax, bx
025     add ax, dx
026     shr ax, 1
027     mov si, ax
028     add si, si
029     cmp cx, ArrList[si]
030     jae greaterEq
031
032     dec ax
033     mov dx, ax
034     jmp mainFunc
035
036 ;if arrList[mid]<=key
037 greaterEq:
038     je ifMatched
039
040     inc ax
041     mov bx, ax
042     jmp mainFunc
043
044 ;if arrList[mid]=key
045 ifMatched:
046     add al, 01
047     add al, 48
048     lea si, Out2
049
050     mov [si], al
051     lea dx, Out1
052     jmp dispOut
053
054 notMatched:
055     lea dx, failOut
056
057 dispOut:
058     mov ah, 09h

```

line: 5 col: 47 drag a file here to open

edit: E:\14th semester\cse 360\project360.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```

01 ;initializing necessary data
02 .data
03 Out1 db "Element Found At Position : "
04 Out2 db " !!$"
05 failOut db "Searched Element Not Found!!$"
06 ArrList dw 12h,28h,100h,212h,228h,534h,550h,888h,980h
07 ArrSz dw ($-ArrList)/2
08 Key equ 534h
09
10 ;starting main code
11 .code
12 main proc
13     mov ax, @data
14     mov ds, ax
15
16     mov bx, 00             ;Lower bound to base reg
17     mov dx, ArrSz         ;Upper bound to data reg
18     mov cx, Key           ;Key to count reg
19 ;main function
20 mainFunc:
21     cmp bx, dx
22     ja notMatched
23
24     mov ax, bx
25     add ax, dx
26     shr ax, 1
27     mov si, ax
28     add si, si
29     cmp cx, ArrList[si]
30     jae greaterEq
31
32     dec ax
33     mov dx, ax
34     jmp mainFunc
35
36 ;if arrList[mid]<=key
37 greaterEq:
38     je ifMatched
39
40     inc ax
41     mov bx, ax
42     jmp mainFunc
43
44 ;if arrList[mid]=key
45 ifMatched:
46     add al, 01
47     add al, 48
48     lea si, Out2
49
50     mov [si], al
51     lea dx, Out1
52     jmp dispOut
53
54 notMatched:
55     lea dx, failOut
56
57 dispOut:
58     mov ah, 09h

```

original source c...

```

01 ;initializing necessary
02 .data
03 Out1 db "Element Found A
04 Out2 db " !!$"
05 failOut db "Searched Ele
06 ArrList dw 12h,28h,100h
07 ArrSz dw ($-ArrList)/2
08 Key equ 534h
09
10 ;starting main code
11 .code
12 main proc
13     mov ax, @data
14     mov ds, ax
15
16     mov bx, 00             ;Low
17     mov dx, ArrSz         ;Upp
18     mov cx, Key           ;Key
19 ;main function

```

emulator: project360.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers	H	L
AX	00	00
BX	00	00
CX	00	A7
DX	00	00
CS	0716	
IP	0000	
SS	0710	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0716:0000

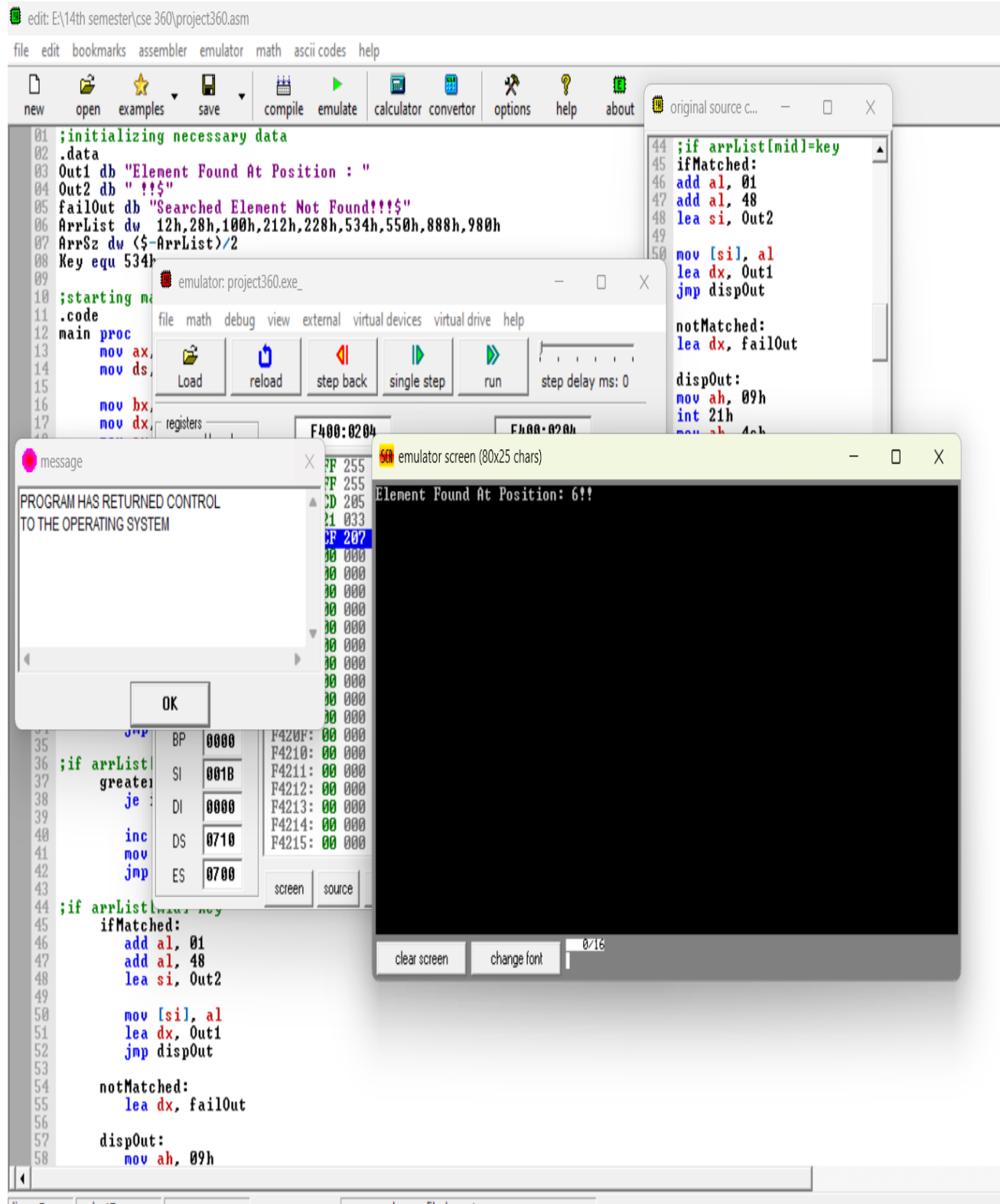
0716:0000

MOV AX, 00710h
MOV DS, AX
MOV BX, 00000h
MOV DX, [0004Fh]
MOV CX, 00534h
CMP BX, DX
JNB 03Ch
MOV AX, BX
ADD AX, DX
SHR AX, 1
MOV SI, AX
ADD SI, SI
CMP CX, [SI] + 03Dh
JNB 027h
DEC AX
MOV DX, AX
JMP 0Fh
JZ 02Eh
INC AX
MOV BX, AX
JMP 0Fh
...

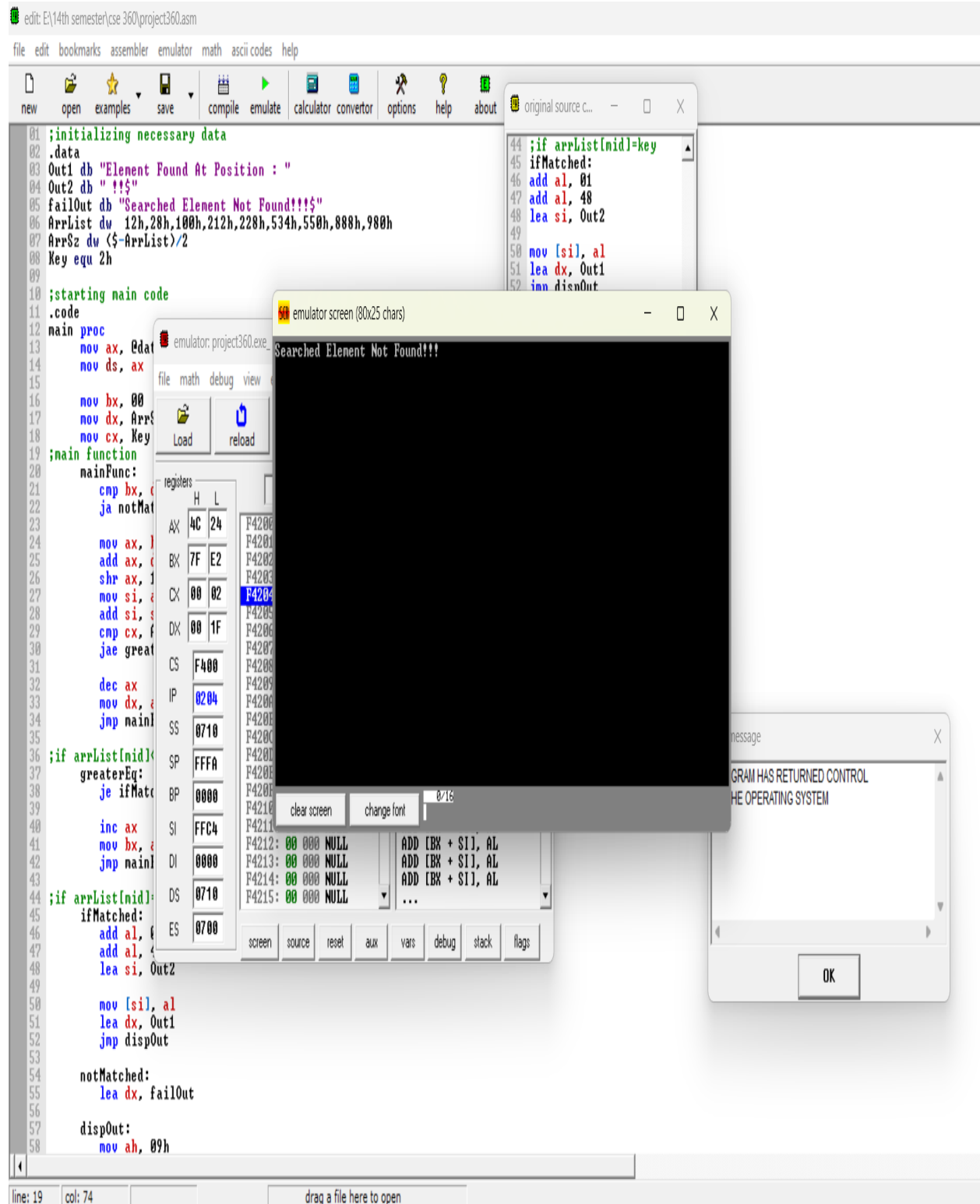
screen source reset aux vars debug stack flags

line: 5 col: 47 drag a file here to open

It is ready to emulate.



Here is the output. We find 534h also show the location which is 6



Here the input is 2h. We searched for it, but didn't find it. And the output is not found.

Conclusion and Future Improvements:

The Binary Search method is effectively implemented using the given 8086 assembly code. It shows the assembly language's efficiency and power in handling low-level tasks like searching in sorted arrays. However, it lacks the stability and error handling required for real-world applications. You may need to improve the code to handle edge cases and offer more complete error checking, depending on your individual use case. Error management, modularization, efficient division for midpoint calculation, understandable output, code comments, and comprehensive testing for enhanced reliability and usability are possible future improvements for the 8086 Assembly Binary Search code.

Bibliography:

1. Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems (4th ed.). Pearson.
2. This book provides insights into the fundamentals of computer architecture and assembly language, which can be helpful for understanding low-level tasks like binary search.
3. Irvine, K. (2018). Assembly Language for x86 Processors (8th ed.). Pearson.
4. A comprehensive textbook on assembly language programming for x86 processors, including the 8086. It can serve as a valuable resource for learning assembly language.
5. AMD. (1988). AMD Am386 Microprocessor Data Sheet. Advanced Micro Devices.
6. This data sheet provides technical information about the 8086's successor, the Am386 microprocessor, which can be useful for understanding the architecture and instructions of the 8086.
7. Intel Corporation. (1989). 8086/8088 User's Manual. Intel Corporation.
8. The official user's manual for the 8086 microprocessor, which contains detailed information about the architecture and instruction set.
9. McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction (2nd ed.). Microsoft Press.
10. This book covers software development best practices, including error handling, modularization, code comments, and testing, which can be applied to assembly language code.
11. Knuth, D. E. (1968). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley.
12. A classic book on computer algorithms and data structures that can provide insights into binary search and its efficiency.
13. Microsoft Developer Network (MSDN) - Assembly Language Programming for x86 Processors
14. Online resource with articles and tutorials on assembly language programming for x86 processors, including examples and best practices.
15. Remember to format your bibliography according to your institution's or publication's specific guidelines (e.g., APA, MLA, Chicago style). Additionally, make sure to include any other sources you may have consulted during your research and writing process.