Assembly coding ABC-4

**Assembler-Linker-Loader**

**1. Assembler**

1.1)) Consider the three 51 source code modules coded in 3 separate files.

|  |  |  |
| --- | --- | --- |
| file1.asm | file2.asm | file3.asm |
| ORG 0000H  mov SP, #60H  mov DPTR, #TASK\_TABLE  start:  mov R3, P0  ; input from P0  ; values: 0-3 guaranteed  clr A  mov R0, A  ; computing the entry-pt. of TASK\_TABLE  comp\_table\_entry1  xch A, R3  cjne A, R0, comp\_table\_entry2  jmp task\_go  comp\_table\_entry2:  xch A, R3  add A, #3  inc R0  jmp comp\_table\_entry1  task\_go:  jmp @A+DPTR  resume:  jmp start  TASK\_TABLE  jmp task1  jmp task2  jmp task3  jmp task4  END  ; (1) what if the range of P0-input may fall  ; beyond [0-3]? Code enhancement??  ; (2) instead of using CJNE-inst.,  ; is DJNZ-instruction applicable? How  ; would you do it? | ORG 0000H  task1:  push R3  mov R3, #11H  mov P1, #22H  call func1  pop R3  jmp resume  task2:  push A  mov A, #11H  call func2  pop A  jmp resume  task3:  mov P2, #33  jmp resume  task4:  mov P3, #55  mov 20H, P3  jmp resume  func1:  push A  mov A, R3  anl A, #55H  mov P1, A  call func11  call func12  pop A  ret  func2:  call func21  call func22  ret  func2211:  push A  push B  mul A, B  mov 50H, B  mov 51H, A  pop B  pop A  div A, B  mov 52H, A  mov 53H, B  ret  END | ORG 0000H  func11:  push A  anl P1, #7FH  mov A, P1  orl P1, #22H  add A, P1  mov 30H, A  pop A  ret  func12:  push A  orl P1, #0CCH  mov A, P1  anl P1, #0FEH  add A, P1  mov 31H, A  call func121  pop A  ret  func121:  push A  mov A, 30H  add A, 31H  mov 32H, A  pop A  ret  func21:  push A  add A, P2  mov 40H, A  pop A  ret  func22:  push A  clr C  sub A, P2  mov 41H, A  call func221  pop A  ret  func221  push A  push B  mov A, 40H  mov B, 41H  call func2211  pop B  pop A  ret  END |

1.2)) assembling (code translation) of file1.c

|  |  |  |
| --- | --- | --- |
| source lines | machine code bytes | resolved and unresolved address |
| ORG 0000H  mov SP, #60H  mov DPTR, #TASK\_TABLE  start:  mov R3, P0  clr A  mov 4FH, A  comp\_table\_entry1  xch A, R3  cjne A, 4FH, comp\_table\_entry2  ajmp task\_go  comp\_table\_entry2:  xch A, R3  add A, #3  inc 4FH  ajmp comp\_table\_entry1  task\_go:  jmp @A+DPTR  resume:  sjmp start  TASK\_TABLE  ljmp task1  ljmp task2  ljmp task3  ljmp task4  END | 0000H: 05h, 81h, 60h  0003H: 90h, GGh, HHh  0006H: ABh, 80h  0008H: E4h  0009H: E8h, 4Fh  000BH: CBh  000CH: B5h, 4Fh, IIh  000FH: JJh, KKh  0011H: CBh  0012H: 24h, 03h  0014H: 05h, 4Fh  0016H: LLh, MMh  0018H: 73h  0019H: 80h, NNh  001CH: 02h, OOh, PPh  001FH: 02h, QQh, RRh  0022H: 02h, SSh, TTh  0025H: 02h, UUh, VVh  0028H: | Note that all code bytes left blank are  addresses associated with ***labels*** (i.e.,  starting address of a targeted instruction).  a] all address in blanks are either relative  or absolute; can you tell which is which?  b] can you determine all the addresses?  As is already noted, some of the ***labels***  can’t find corresponding addresses when  ***file1.c*** is assembled alone (the so called  ***unresolved labels***).  a] can you tell which ones?  b] why is that?  c] how do you think the problem may be tackled? |

1.3)) assembling (code translation) of file2.c

|  |  |  |
| --- | --- | --- |
| source lines | machine code bytes | resolved and unresolved address |
| ORG 0000H  task1:  push R3  mov R3, #11H  mov P1, #22H  call func1  pop R3  ljmp resume  task2:  push A  mov A, #11H  call func2  pop A  ljmp resume  task3:  mov P2, #33  jmp resume  task4:  mov P3, #55  mov 20H, P3  jmp resume  func1:  push A  mov A, R3  anl A, #55H  mov P1, A  call func11  call func12  pop A  ret  func2:  call func21  call func22  ret  func2211:  push A  push B  mul A, B  mov 50H, B  mov 51H, A  pop B  pop A  div A, B  mov 52H, A  mov 53H, B  ret  END | 0000H: C0h, 03h  0002H: 7Bh, 11h  0004H: 75h, 90h, 22h  0007H: 12h, GHh, HIh  000AH: D0h, 03h  000CH: 02h, IJh, JKh  000FH: C0h, E0h  0011H: 74h, 11h  0013H: 12h, KLh, LMh  0016H: C0h, E0h  0018H: 02h, MNh, NOh  001BH: 75h, A0h, 21h  001EH: 02, OPh, PQh  0021H: 75h, B0h, 37h  0024H: 85h, 20H, B0h  0027H: 02h, QRh, RSh  002AH: C0h, E0h  002CH: EBh  002DH: 54h, 55h  002FH: F5h, 90h  0031H: 12h, STh, TUh  0034H: 12h, UVh, VWh  0037H, D0h, E0h  0039H: 22h  003AH: 12h, WXh, XYh  003DH: 12h, YZh, ZGh  0040H: 22h  0041H: C0h, E0h  0043H: C0h, F0h  0045H: A4h  0046H: 85h, 50h, F0h  0049H: F5h, 51h  004BH: D0h, F0h  004DH: D0h, E0h  004FH: 84h  0050H: F5h, 52h  0052H: 85h, 53h, F0h  0055H: 22h  0056H | Note that all code bytes left blank are  addresses associated with ***labels*** (i.e.,  starting address of a targeted instruction).  a] all address in blanks are either relative  or absolute; can you tell which is which?  b] can you determine all the addresses?  As already noted, some of the ***labels*** can’t  find corresponding addresses when ***file2.c***  is assembled alone (the so called  ***unresolved labels***).  a] can you tell which ones?  b] why is that?  c] how do you think the problem may be tackled? |

1.4)) assembling (code translation) of file3.c

|  |  |  |
| --- | --- | --- |
| source lines | machine code bytes | resolved and unresolved address |
| ORG 0000H  func11:  push A  anl P1, #7FH  mov A, P1  orl P1, #22H  add A, P1  mov 30H, A  pop A  ret  func12:  push A  orl P1, #0CCH  mov A, P1  anl P1, #0FEH  add A, P1  mov 31H, A  call func121  pop A  ret  func121:  push A  mov A, 30H  add A, 31H  mov 32H, A  pop A  ret  func21:  push A  add A, P2  mov 40H, A  pop A  ret  func22:  push A  clr C  subb A, P2  mov 41H, A  call func221  pop A  ret  func221  push A  push B  mov A, 40H  mov B, 41H  call func2211  pop B  pop A  ret  END | 0000H: C0h, E0h  0002H: 53h, 90h, 7Fh  0005H: E5h, 90h  0007H: 43h, 90h, 22h  000AH: 25h, 90h  000CH: F5h, 30h  000EH: D0h, E0h  0010H: 22h  0011H: C0h, E0h  0013H: 43h, 90h, CCh  0016H: E5h, 90h  0018H: 53h, 90h, FEh  001BH: 25h, 90h  001DH: F5h, 31h  001FH: 12h, GIh, HJh  0022H: D0h, E0h  0024H: 22h  0025H: C0h, E0h  0027H: E5h, 30h  0029H: 25h, 31h  002BH: F5h, 32h  002DH: D0h, E0h  002FH: 22h  0030H: C0h, E0h  0032H: 25h, A0h  0034H, F5h, 40h  0036H: D0h, E0h  0038H: 22h  0039H: C0h, E0h  003BH: C3h  003CH: 95h, B0h  003EH: F5h, 41h  0040H: 12h, IKh, JLh  0043H: D0h, E0h  0045H: 22h  0046H: C0h, E0h  0048H: C0, F0h  004AH: E5h, 40h  004CH: 85h, F0h, 41h  004FH: 12h, KMh, LNh  0052H: D0h, F0h  0054H: D0h, E0h  0056H: 22h  0057H: | Note that all code bytes left blank are  addresses associated with ***labels*** (i.e.,  starting address of a targeted instruction).  a] all address in blanks are either relative  or absolute; can you tell which is which?  b] can you determine all the addresses?  As already noted, some of the ***labels*** can’t  find corresponding addresses when ***file3.c***  is assembled alone (the so called  ***unresolved labels***).  a] can you tell which ones?  b] why is that?  c] how do you think the problem may be tackled? |

1.5)) **object files:** file1.asm 🡪 file1.obj file2.asm 🡪 file2.obj file3.asm 🡪 file3.obj

|  |  |  |  |
| --- | --- | --- | --- |
| file1.obj | file2.obj | file3.obj | cross reference of labels |
| code-table;  code-size: 40 bytes   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***start***  ***task\_go***  ***resume***  ***TASK\_TABLE***  ***comp\_...entry1***  ***comp\_...entry2*** | 0006H  0018H  0019H  001CH  000BH  0011H | | unresolved-name | address | | ***task1***  ***task2***  ***task3***  ***task4*** | \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\* | | code-table;  code-size: 86 bytes   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***task1***  ***task2***  ***task3***  ***task4***  ***func1***  ***func2***  ***func2211*** | 0000H  000FH  001BH  0021H  002AH  003AH  0041H | | unresolved-name | address | | ***resume***  ***func11***  ***func12***  ***func21***  ***func22*** | \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\* |   - | code-table;  code-size: 87 bytes   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***func11***  ***func12***  ***func121***  ***func21***  ***func22***  ***func221*** | 0000H  0011H  0025H  0030H  0039H  0046H | | unresolved-name | address | | ***func2211*** | \*\*\*\* | | a] code modules concurrently and separately programmed in multiple files are common practices during system development, as shown in this particular example;  b] as a result, referencing to modules defined in different  source files occurs quite often, so is branching to labels defined elsewhere; as demonstrated in this example (the label-cross-referencing as so called)  c] as a result, access to variable names defined in other source files are inevitable (data-cross-referencing )not addressed in this example) |

2. **Linker**

2.1)) **object files linking:** file1.obj -- file2.obj -- file3.obj 🡪 ***linked\_module.obj***

|  |  |  |  |
| --- | --- | --- | --- |
| file1.obj | file2.obj | file3.obj | object files linkage |
| code-table;  code-size: 40 bytes;   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***start***  ***task\_go***  ***resume***  ***TASK\_TABLE***  ***comp\_...entry1***  ***comp\_...entry2*** | 0006H  0018H  0019H  001CH  000BH  0011H | | unresolved-name | address | | ***task1***  ***task2***  ***task3***  ***task4*** | \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\* | | code-table;  code-size: 86 bytes;   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***task1***  ***task2***  ***task3***  ***task4***  ***func1***  ***func2***  ***func2211*** | 0000H  000FH  001BH  0021H  002AH  003AH  0041H | | unresolved-name | address | | ***resume***  ***func11***  ***func12***  ***func21***  ***func22*** | \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\* |   - | code-table;  code-size: 87 bytes;   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***func11***  ***func12***  ***func121***  ***func21***  ***func22***  ***func221*** | 0000H  0011H  0025H  0030H  0039H  0046H | | unresolved-name | address | | ***func2211*** | \*\*\*\* | | ***LINKER*** is the system module that organizes code bytes in the 3 object files as one, in the order as specified  by the obj.-file sequence.  Can you tell  a] the code size of  ***linked\_module.obj***?  b] the starting address of ***task1***?  c] the starting address of ***func11?***  d] the addresses of all ***labels***? |

2.2)) **object files linking:** file1.obj -- file2.obj -- file3.obj 🡪 ***linked\_module.obj***

|  |  |  |  |
| --- | --- | --- | --- |
| file1.obj | file2.obj | file3.obj | object files linkage |
| code-table;  code-size: 40 bytes;   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***start***  ***task\_go***  ***resume***  ***TASK\_TABLE***  ***comp\_...entry1***  ***comp\_...entry2*** | 0006H  0018H  0019H  001CH  000BH  0011H | | unresolved-name | address | | ***task1***  ***task2***  ***task3***  ***task4*** | \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\* | | code-table;  code-size: 86 bytes;   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***task1***  ***task2***  ***task3***  ***task4***  ***func1***  ***func2***  ***func2211*** | ??? | | unresolved-name | address | | ***resume***  ***func11***  ***func12***  ***func21***  ***func22*** | \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\*  \*\*\*\* |   - | code-table;  code-size: 87 bytes;   |  |  | | --- | --- | | symbol-table | | | resolved-name | address | | ***func11***  ***func12***  ***func121***  ***func21***  ***func22***  ***func221*** | ??? | | unresolved-name | address | | ***func2211*** | \*\*\*\* | | ***LINKER*** is the system  module that organizes code bytes in the 3 object files as one, in the order as specified  by the obj.-file sequence.  If file2.asm and file3.asm starts with ORG 0100H and  ORG 0200H, respectively,  can you tell  a] what the symbol-table in file2.obj will be?  b] what the symbol-table in file3.obj will be?  c] the code size of  ***linked\_module.obj***?  d] the starting address of ***task1***?  e] the starting address of ***func11?***  f] the addresses of all ***labels***? |

2.3)) contents of ***linked\_module.obj*** by the linker

|  |  |
| --- | --- |
| linked\_module.obj in the case of 2.1)) | object files linkage |
| code-table;  code-size: 0D5H (213) bytes;   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | symbol-table | | | | | | | label-name | address | label-name | address | resolved-name | address | | ***start***  ***task\_go***  ***resume***  ***TASK\_TABLE***  ***comp\_...entry1***  ***comp\_...entry2*** | 0006H  0018H  0019H  001CH  000BH  0011H | ***task1***  ***task2***  ***task3***  ***task4***  ***func1***  ***func2***  ***func2211*** | 0028H  0037H  0043H  0049H  0052H  0062H  0069H | ***func11***  ***func12***  ***func121***  ***func21***  ***func22***  ***func221*** | 007EH  008FH  00A3H  00AEH  00B7H  00C4H |   in the code-table, try determining the following address-related code bytes  \* GGh:=? HHh:=? IIh:=? . . . UUh:=? VVh:=?  \* GHh:=? HIh:=? . . . YZh:=? ZGh:=?  \* GIh:=? HJh:=? . . . KMh:=? MOh:=? | ***LINKER*** is the system module that organizes code bytes in the 3 object files as one, in the order as specified  by the obj.-file sequence.  Can you tell  a]how the code-table may look like?  b] why the values of ***task1*** and ***func11*** are respectively as such?  c] addresses for all ***label-names***? |

2.4)) contents of ***linked\_module.obj*** by the linker

|  |  |
| --- | --- |
| linked\_module.obj in the case of 2.2)) | object files linkage |
| code-table;  code-size: 0D5H (213) bytes;   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | symbol-table | | | | | | | label-name | address | label-name | address | resolved-name | address | | ***start***  ***task\_go***  ***resume***  ***TASK\_TABLE***  ***comp\_...entry1***  ***comp\_...entry2*** | 0006H  0018H  0019H  001CH  000BH  0011H | ***task1***  ***task2***  ***task3***  ***task4***  ***func1***  ***func2***  ***func2211*** | 0100H  010FH  011BH  0121H  012AH  013AH  0141H | ***func11***  ***func12***  ***func121***  ***func21***  ***func22***  ***func221*** | 0200H  0211H  0225H  0230H  0239H  0246H |   in the code-table, try determining the following address-related code bytes  \* GGh:=? HHh:=? IIh:=? . . . UUh:=? VVh:=?  \* GHh:=? HIh:=? . . . YZh:=? ZGh:=?  \* GIh:=? HJh:=? . . . KMh:=? MOh:=? | ***LINKER*** will be the system module that organizes code bytes in the 3 object files as one, in the order as specified  by the obj.-file sequence.  Can you tell  a]how the code-table may look like?  b] why the values of ***task1*** and ***func11*** are respectively as such?  c] addresses for all ***label-names***? |

3. **LOADER**

3.1)) ***loader*** is system module that loads the objet-module generated by the ***linker*** at a specified location in code-space.

3.2)) Consider the ***linked\_module.obj***s respectively in 2.3)) and 2.4)). When either one is loaded at **0000H** of the built-in

51-code-memory, an executable code image is formed in the code-space accordingly.

\* can you see that code bytes in *2.3))****linked\_module.obj*** (213 bytes in length) is exactly the same as the corresponding code image (213 bytes in length starting from 000H)?

\* can you SEE how the code image of *2.4))****linked\_module.obj*** may look like? Note that the code image also starts from 0000H, and ends at 0257H, with two “gap”s respectively between 0028H and 00FFH, and between 0156H and 01FFH?

|  |
| --- |
| 05H |
| . . . |
| C0H |
| . . . |
| C0H |
| . . . |
|  |

|  |
| --- |
| 05H |
| . . . |
|  |
| C0H |
| . . . |
|  |
| C0H |
| . . . |
|  |

**0000H**: **0000H**:

0028H: 0028H:

007EH: 00FFH:

0100H:

00D5H:

0156H:

0FFFH:

01FFH:

0200H:

0257H:

0FFFH:

[code-image of *2.3))****linked\_module.obj***] [code-image of *2.4))****linked\_module.obj***]

3.3)) Consider the ***linked\_module.obj***s respectively in 2.3)) and 2.4)). When either one is loaded ata location being different from the one assumed in the ***linke***r, say **0020H,** of built-in 51-code-memory, an executable code image is formed in the code-space with all code bytes associated with absolute addressing being modified accordingly by the ***loader.***

\* can you see that code bytes in *2.3))****linked\_module.obj*** (213 bytes in length) is exactly the same as the corresponding code image (213 bytes in length starting from 0020H)?

\* can you SEE how the code image of *2.4))****linked\_module.obj*** may look like? Note that the code image also starts from 0020H, and ends at 0257H, with two “gap”s respectively between 0048H and 011FH, and between 0126H and 021FH?

|  |  |  |
| --- | --- | --- |
| code-image of *2.3))****linked\_module.obj*** |  | code-image of *2.3))****linked\_module.obj*** |
| |  |  | | --- | --- | | 0000H: |  | | 0020H: | 05H | | . . . | | 0048H: | C0H | | . . . | | 009EH: | C0H | | . . . | | 00F5H:  0FFFH: |  | |  | |  |  | | --- | --- | | 0000H: |  | | 0020H: | 05H | | . . . | | 0048H:  011FH: |  | | 0120H: | C0H | | . . . | | 0176H:  021FH: |  | | 0220H: | C0H | | . . . | | 0277H:  0FFFH: |  | |