**Assembly Programming 51-sample[B]**

operations on data blocks

[1] Consider two data blocks, each being 8 bytes in size, residing in the built-in 51 data space respectively at 40H and 50H. The two blocks are to be merged as a 16-byte one, in a shuffling manner as depicted below. The resultant block starts at 60H.

40H 50H

47H 57H

60H

. . .

. . .

6FH

ORG 0

mov 58H, #40H

mov 59H, #50h

mov R1, #60H

mov R3, #8

next: mov R0, 58H

mov A, @R0

mov @R1, A

inc R1

mov R0, 59H

mov A, @R0

mov @R1, A

inc R1

inc 58H

inc 59H

djnz R3, next

END

[2] Consider a 16-byte data block residing at 30H of the built-in 51 data space, which is now to be spited into two 8-byte data blocks, residing respectively at 40H and 50H.

30H even-addressed

31H odd-addressed

. . .

37H

. . .

3FH

40H 50H

. . . . . .

47H 57H

ORG 0

mov 60H, #40H

mov 61H, #50H

mov R0, #30H

mov R3, #8

next: mov A, @R0

mov R1, 60H

mov @R1, A

inc R0

mov A, @R0

mov R1, 61H

mov @R1, A

inc R0

inc 60H

inc 61H

djnz R3, next

END

[3] Consider two data blocks, each holding 4 16-bit unsigned integers , reside in the built-in 51 data space respectively at 40H and 50H. The two blocks are to be merged as one holding 8 16-bit unsigned integers, in a shuffling manner as depicted below. The resultant block starts at 60H.

40H 50H

42H 52H

44h 54H

46H 56H

60H

62H

64H

66H

. . .

6CH

6EH

ORG 0

mov 30H, #40H

mov 31H, #50H

mov R1, #60H

mov R3, #4

next:

mov R0, 30H

mov A, @R0

mov @R1, A

inc R1

inc R0

mov A, @R0

mov @R1, A

inc R1

inc R0

mov 30H, R0

mov R0, 31H

mov A, @R0

mov @R1, A

inc R1

inc R0

mov A, @R0

mov @R1, A

inc R1

inc R0

mov 31H, R0

djnz R3, next

END

[4] Consider a 64-bit unsigned integer residing at 30H of the 51 built-in data space, in little-endian order.

The same integer is to be saved at 40H, in big-endian order, as done by the code lines given below.

ORG 0

mov R0, #37H

mov R1, #40H

mov R3, #8

next:

mov A, @R0,

mov @R1, A

dec R0

inc R1

djnz R3, next

END

[alternative 1 ]

ORG 0

mov SP, #60H

push 30H

push 31H

push 32H

push 33H

push 34H

push 35H

push 36H

push 37H

pop 40H

pop 41H

pop 42H

pop 43H

pop 44H

pop 45H

pop 46H

pop 47H

END

[alternative 2 ]

ORG 0

mov SP, #60H

mov R0, #30H

mov R1, #40H

mov R3, #8

next1:

mov A, @R0

push A

inc R0

djnz R3, next1

mov R3, #8

next2:

pop A

mov @R1, A

inc R1

djnz R3, next2

END

[5] Consider a 256-bit unsigned integer residing, in little-endian order, at 30H of 51 built-in data space. How the task of order conversion to big-endian using the same data space could be done?

[hint]

A minor modification to the code lines given in [4] [alternative 2] would suffice.

[6] Consider two data blocks of 16-bit unsigned integers, A and B, residing in 51 built-in data space. A starts at 30H with 4 integers and B starts at 40H with 8 integers. How the task of forming a new block, C, of 12 integers at 50H, by inserting each entity of A sequentially into the locations following every two entities of B, could be done is as shown below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

A:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

B:

ORG 0

mov 10H, #30H

mov 11H, #40H

mov R1, #60H

mov R3, #4

next:

push 3 ; push R3

mov R3, #4

next1:

mov R0, 10H

mov A, @R0

mov @R1, A

inc 10H

inc R1

djnz R3, next1

mov R3, #2

next2:

mov R0, 11H

mov A, @R0

mov @R1, A

inc 11H

inc R1

djnz R3, next2

pop 3 ; pop R3

djnz R3, next

END

[further pushing]

What if the block C is to be placed at 50H of the 51 built-in data space (block B thus overwritten)?

[7] Consider a data block of 12 16-bit unsigned integers, A, starting at 30H of the 51 built-in data space.

[code specifications]

\* A new data block of 3 integers, B, is to be established at 50H by extracting the last entity from every 4 integers of A.

\* A stays in the same data space, leaving no vacancies after extraction.

A:

30H 47H

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |

B:

|  |  |  |
| --- | --- | --- |
|  |  |  |

A’:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |

30H 40H

code sample:

ORG 0

mov R1, #50H

mov 70H, #30H

mov SP, #6FH

mov A, 70H

add A, #6

mov 71H, A

mov R3, #3

next4:

mov R0, 70H

cjne A, R0, skip3

mov R2, #2

extract:

mov A, @R0

mov @R1, A

inc R1

inc R0

djnz R2, extract

mov 70H, R0

mov A, 71H

add A, #8

mov 71H, A

djnz R3, next4

jmp cont

skip3:

mov R2, #6

skip:

inc 70H

push A

djnz R2, skip

mov A, 71H

jmp next4

cont:

mov R1, #3FH

mov R3, #16

restore:

pop A

mov @R1, A

dec R1

djnz R3, restore

END

[further push]

The code lines in the sample code is not well structured, leading to a somewhat messy looking of the code. Try if you could reorganize it for a neater and cleaner look.

ORG 0

code lines

of

your own

END