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

Addition for 64-bit unsigned integers

108-01-31

The performance of a program can be evaluated by certain numbers of indices; the quality of which relies on, largely, the programmer’s experiences.

Consider the task of adding up two 64-bit unsigned integers, for which several 51-assembly programs are given below for one’s reference.

Suppose the two integers are stored in the built-in data space of the 51µp, respectively at addresses 30H and 38H, in little-endian order. The 64-bit sum is to be saved at the address 50H.

msB lsB

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

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

= = = = = = = = = = = = = = = = = = = = = = = =

SAMPLE [1]

= = = = = = = = = = = = = = = = = = = = = = = = =

ORG 0

|  |
| --- |
| mov A, 30H  add A, 38H  mov 50H, A  mov A, 31H  addc A, 39H  mov 51H, A  mov A, 32H  addc A, 3AH  mov 52H, A  mov A, 33H  addc A, 3BH  mov 53H, A  mov A, 34H  addc A, 3CH  mov 54H, A  mov A, 35H  addc A, 3DH  mov 55H, A  mov A, 36H  addc A, 3EH  mov 56H, A  mov A, 37H  addc A, 3FH  mov 57H, A |

END

**[notes]**

\* the instruction lines marked in the block is referred to as the code-body;

\* ***mov A, nnH***

the instruction moving a byte of data from the address nnH(00-FF) of the 51 built-in data space to ***A***, the ***accumulator*** in 51µp;

\* ***add A, mmH***

the instruction adds up 2 operands, one being the byte retrieved from address mmH of the 51 built-in data space and another the byte in the accumulator A;

the sum is then saved in the accumulator A

\* ***addc A, mmH***

the addition by this instruction involves 3 operands: A, (mmH) and the ***CARRY***, a bit-status (0 or 1) resulted from previous arithmetic-ops or carry-bit associated operations;

\* ***mov xxH, A***

the instruction moves a byte of data in A to the byte at the address xxH(00-FF) of the 51 built-in data space;

\* ***ORG***/***END***

known as the ***directive***s supported by the 51-assembler, with which the program may “direct” the assembler for certain specific tunings while translating the code lines;

***ORG*** specifies the start-address of the translated code

***END*** tells 51-assembler that the end of source lines is reached

**[observations]**

\* the code is a bit too lengthy(considering

the case if 128-bit addition is required);

a somewhat more compact coding style

is preferred;

\* code lines for each of the 8 add-ops look

“similar”;

\* values of the two operands unknown;

\* can you “perceive” how these three 8-byte operands involved in the program

are arranged in the 256-byte built-in data space of the 51µp?

= = = = = = = = = = = = = = = = = = = = = = = =

SAMPLE [2]

= = = = = = = = = = = = = = = = = = = = = = = = =

ORG 0

mov 40H, #30H

mov 41H, #38H

mov 42H, #50H

mov R3, #8

; code initiation done

clr C

next:

mov R0, 40H

mov R1, 41H

mov A, @R0

addc A, @R1

inc R0

inc R1

mov 40H, R0

mov 41H, R1

mov R0, 42H

mov @R0, A

inc R0

mov 42H, R0

djnz R3, next

; a looping structure for handling 8

; rounds of byte-addition;

;

;[notes]

; **R0**: acting 2 roles in the process

; 1) as the index for sequentially

; retrieving 8 subparts of the

; 1st opnd in each iteration;

; 2) as the index for sequentially

; pointing to 8 subparts of the

; sum-opnd in each iteration;

; **R1**: index to the 2nd opnd for the

; addion

; **40H**: the memory location in

; 51µp built-in data space

; with address 40H;

; purpose?

; **41H**: data-memory at 41H-addr;

; purpose?

; **42H**: data-memory at 42H-addr;

; purpose?

; **#30H**, **#38H**, **#50H**:

; hex-value led by a pound-sign for

; specifying 3 values immediately

; to be exploited in the code

; **C**: the bit carry-status in 51µp

; architecture

; **djnz**: decrease then jump on nonzero

; “djnz counter, label”

; the conditional branching (jump)

; instruction for looping control in code

; body with a down-counter (here R3)

; 1] counter--

; 2] if counter!-=0

; jump to the labeled-inst.

; else

; execute the inst.

; immediately following

; djnz-inst.;

;

; **next:** the label as the jump-target of

; djnz-inst,;

END

[**observations**]

\* coding looks not so straight forward as

the one given in 1];

\* coding looks more compact and elegant as compared to the one in sample [1];

\* ***implication of @R0 and @R1??***

\* possibility for further compactness?

\* still, values of the 2 operands remain unspecified

= = = = = = = = = = = = = = = = = = = = = = = =

SAMPLE [3]

= = = = = = = = = = = = = = = = = = = = = = = = =

ORG 0

mov 40H, #30H

;;; mov 41H, #38H

***mov R1, #38H***

mov 42H, #50H

mov R3, #8

; code initiation done

clr C

next:

mov R0, 40H

;;; mov R1, 41H

mov A, @R0

addc A, @R1

inc R0

inc R1

mov 40H, R0

;;; mov 41H, R1

mov R0, 42H

mov @R0, A

inc R0

mov 42H, R0

djnz R3, next

END

[**notes**]

\* reduction of code lines:

- the lines marked with ;;; are omitted

- 1 new code line needed (in *italics*)

- data space at 41H spared

\* 1 byte less in memory usage;

[**observations**]

\* pushing for further compactness??

\* still, values of the 2 operands remain unspecified

= = = = = = = = = = = = = = = = = = = = = = = =

SAMPLE [4]

= = = = = = = = = = = = = = = = = = = = = = = = =

org 0 ; case-insensitive

; assembler

mov 40H, #30H

mov 42H, #50H

***mov R1, #38H***

mov R3, #8

; code initiation done

mov A, #0

mov R0, #30H

mov r3, #10H

nextt: mov @R0, A

inc R0

inc A

djnz R3, nextt

; initial values for the two

; operands done

clr C

next:

mov R0, 40H

mov A, @R0

addc A, @R1

inc 40H

inc R1

mov R0, 42H

mov @R0, A

inc 42H

djnz R3, next

end

[**observations**]

\* can you SEE the values of the 2

operands set prior to the 8-round

add-process? And then predict the sum at 50H?

\* code readability looks poor, implying hard time in following up code maintenance;