In Program 1, there are 32 instructions, starting from 0000, and 4 initial data in 0064, 007C, 007D, 007F which are location 100, 124, 125, 127. All the numbers for program1 in this simulator are assumed as unsigned number.

The index register 2 is 100, so instruction can access the data area starting from 100 to 127.

Below, nums[1..20] is the list of 20 numbers, target is the number to compare, and i is used for counting the loop and computing the location of each nums[i]. i is initialized as 20 and finally is 0 as the program decrements through the list.

Data:

The General purpose registers are used as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | 00 | 01 | 10 | 11 |
| Content | result | EA for 20 numbers | Place for compare and subtract | i |

Memory data area:

|  |  |
| --- | --- |
| Location | Content |
| 100 | i for computing(initialized as 20) |
| 101 | nums[1] |
| ... | ... |
| 120 | nums[20] |
| 121 | result |
| 122 | target |
| 123 | EA for nums[i] |
| 124 | min (the smallest difference now, initialized as FFFF) |
| 125 | 121 (used for computing the EA: 121 - i = EA) |
| 126 | difference (one loop) |
| 127 | i for storing 20 numbers(initialized as 20) |

Instructions:

The instructions can be divided to 3 parts: store 20 numbers, store the compare number, and compute.

Before the loop for storing 20 numbers, load the number 20 (i) in location 127 to R3. Then use SOB to loop and each time make i subtract 1 until i becomes 0. In each single loop, the number i is changing and always updating in location 127, so it can be used to compute the EA for nums[i] as 121 - i = EA. Then, store the EA to location 123, so that the same instruction can store all of thes 20 numbers to the correct location from 101 to 120 using indirect location. Storing the target number only needs two instructions.

Before the loop for comparing 20 numbers to the target number, it also load the number 20 (i) first, but because the i in 127 has been updated to 0, it need another i in location 100 and load it to R3. This i is also used for computing EA and enabling the loop using SOB. Each time it compare the nums[i] and target, compute the abs(nums[i] - target) first, and then compare the difference with min in memory location. If the difference is smaller than min, then it update difference to min and store the current nums[i] to result (memory location 121).

To get abs(nums[i] - target) and compare difference and min, it use SMR and then JCC(checking underflow) and JMA to compute. Then there are two branches to deal with the conditions num[i] > target or target > num[i] and difference > min or min > difference.

**The detail of instructions:**

Opcode GPR IX I address

**Storing the 20 numbers**

1) 0000

**LDR mem(127) to r(3) // load i**

000001 11 10 0 11011

0000 0111 1001 1011

079B

2) 0001

**IN devid(0) to r(2) // store num[i] to R2**

110001 10 000 00000

1100 0110 0000 0000

C600

3) 0002

**STR r(3) to mem(127) // store i**

000010 11 10 0 11011

0000 1011 1001 1011

0B9B

4 ) 0003

**LDR mem(125) to R1 // load 121**

000001 01 10 0 11001

0000 0101 1001 1001

0599

5 ) 0004

**SMR R1 - mem(127) to R1 // 121 - i = EA**

000101 01 10 0 11011

0001 0101 1001 1011

159B

6 ) 0005

**STR R1 to mem(123) // store EA**

000010 01 10 0 10111

0000 1001 1001 0111

0997

7) 0006

**STR R2 to mem(mem(123)) // store num[i]**

000010 10 10 1 10111

0000 1010 1011 0111

0AB7

8) 0007

**SOB R3 go to 01**

001110 11 00 0 00001

0011 1011 0000 0001

3B01

**Storing the compare number**

9) 0008

**IN devid(0) to r(2) // store compare number to R2**

110001 10 000 00000

1100 0110 0000 0000

C600

10) 0009

**STR R2 to mem(122) // store compare number to 122**

000010 10 10 0 10110

0000 1010 1001 0110

0A96

**Computing**

11) 000A

**LDR mem(100) to R3 // load i to r3**

000001 11 10 0 00000

0000 0111 1000 0000

0780

12) 000B

**STR R3 to mem(100)    // store i**

000010 11 10 0 00000

0000 1011 1000 0000

0B80

13) 000C

**LDR mem(122) to R2 // load target**

000001 10 10 0 10110

0000 0110 1001 0110

0696

14) 000D

**LDR mem(125) to R1 // load 121**

000001 01 10 0 11001

0000 0101 1001 1001

0599

15) 000E

**SMR R1 - mem(100) to R1 // 121 - i = EA**

000101 01 10 0 00000

0001 0101 1000 0000

1580

16) 000F

**STR R1 to mem(123) // store EA**

000010 01 10 0 10111

0000 1001 1001 0111

0997

17) 0010

**SMR R2 - mem(mem(123)) // target - num[i]**

000101 10 10 1 10111

0001 0110 1011 0111

16B7

18) 0011

**JCC to 13 // if target < nums[i] go to 13**

001010 01 00 0 10011

0010 1001 0001 0011

2913

19) 0012

**JMA to 15 // skip the branch to 15**

001011 00 00 0 10101

0010 1100 0001 0101

2C15

20) 0013

**LDR mem(mem(123)) to R2 // load nums[i]**

000001 10 10 1 10111

0000 0110 1011 0111

06B7

21) 0014

**SMR R2 - mem(122) to R2 // nums[i] - target**

000101 10 10 0 10110

0001 0110 1001 0110

1696

22) 0015

**STR R2 to mem(126) // store differ**

000010 10 10 0 11010

0000 1010 1001 1010

0A9A

23) 0016

**SMR R2 - mem(124) // differ-min**

000101 10 10 0 11000

0001 0110 1001 1000

1698

24) 0017

**JCC to 19 // if differ < min go to 19**

001010 01 00 0 11001

0010 1001 0001 1001

2919

25) 0018

**JMA to 1D // skip the branch to 1B**

001011 00 00 0 11101

0010 1100 0001 1101

2C1D

26) 0019

**LDR mem(126) to R2 // load differ**

000001 10 10 0 11010

0000 0110 1001 1010

069A

27) 001A

**STR R2 to mem(124) // store differ to min**

000010 10 10 0 11000

0000 1010 1001 1000

0A98

28) 001B

**LDR mem(mem(123)) to R2 // load nums[i]**

000001 10 10 1 10111

0000 0110 1011 0111

06B7

29) 001C

**STR R2 to mem(121) // store result**

000010 10 10 0 10101

0000 1010 1001 0101

0A95

30) 001D

**SOB R3 go to 0B // loop**

001110 11 00 0 01011

0011 1011 0000 1011

3B0B

31) 001E

**LDR mem(121) to R0 // load result**

000001 00 10 0 10101

0000 0100 1001 0101

0495

32) 001F

**OUT output r(0) to devid(1)**

110010 00 000 00001

1100 1000 0000 0001

C801