Lecture 12

Control Flow III

How to Avoid Spaghetti Code?

Last Time: `if-else`

Task:

1. Create an array in RAM with values {1, 1, 2, 3, 5, 8, 13, 21}
2. Define two variables `even_sum` and `odd_sum` in RAM
3. Loop trough the array and find the sum of even and odd numbers

This is an `if-else` problem

```
if (element x is even)
{
   even_sum += x;
}
else
{
   odd_sum += x;
}
...
```

How do we check if a number is even?

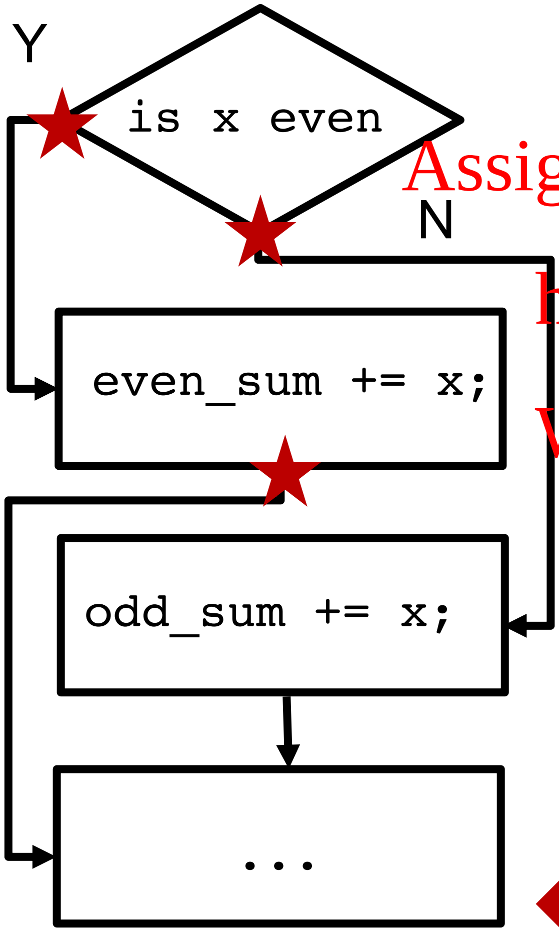
```
bit.w   #BIT0, x
```

If `BIT0` is set (i.e., 1) then the carry bit in SR will be set

```
jc / jnc
```

How to implement `if-else`?

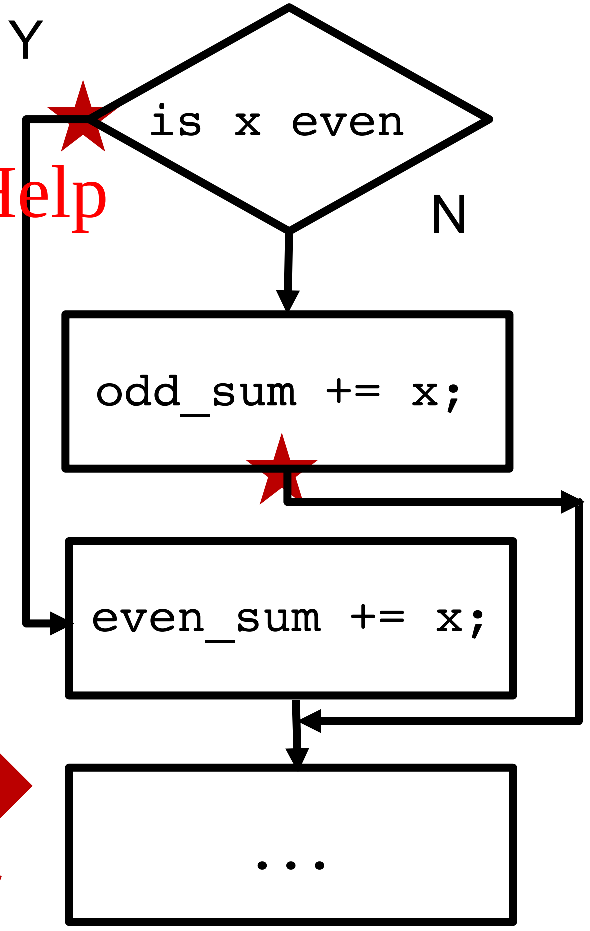
Better to change the order of the blocks



```
if (x is even)
{
    even_sum += x;
}
else
{
    odd_sum += x;
}
```

NOT good

better



Good Implementation of `if-else`

Definitions

```
            .data
            .retain
            .retainrefs

even_sum:   .word   0
odd_sum:    .word   0

array:      .word 1, 1, 2, 3, 5, 8, 13, 21
LENGTH:     .set  16                ; LENGTH of array in bytes
```

Code

```
            .text
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
                clr.w   R4                  ; R4 serves as index, start at 0
                                            ; indices are 0, 2, ..., LENGTH - 2
read:           mov.w   array(R4), R5       ; read array(R4)
                bit.w   #BIT0, R5           ; check least significant bit
                jc      odd                 ; Carry set if bit is 1, i.e., odd number

even:           add.w   R5, even_sum        ; we are here if array(R4) is even
                jmp     proceed             ; proceed index to next element

odd:            add.w   R5, odd_sum         ; we are here if array(R4) is odd


proceed:        incd.w  R4                  ; index points to next element
                cmp.w   #LENGTH, R4         ; check array boundary
                jlo     read                ; break if LENGTH > index

main:           jmp     main
                nop
```

Not so Good Implementation

```
            .data
            .retain
            .retainrefs

even_sum:   .word   0
odd_sum:    .word   0

array:      .word 1, 1, 2, 3, 5, 8, 13, 21
LENGTH:     .set  16                          ; LENGTH of array in bytes

            .text
```

NOT good

```
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
            clr.w   R4                  ; R4 serves as index, start at 0
                                        ; indices are 0, 2, ..., LENGTH - 2
read:       mov.w   array(R4), R5       ; read array(R4)
            bit.w   #BIT0, R5           ; check least significant bit
            jnc     even                ; carry set if bit is 1, i.e., odd number
            jc      odd

even:       add.w   R5, even_sum        ; we are here if array(R4) is even
            jmp     proceed             ; proceed index to next element

odd:        add.w   R5, odd_sum         ; we are here if array(R4) is odd


proceed:    incd.w  R4                  ; index points to next element
            cmp.w   #LENGTH, R4         ; check array boundary
            jlo     read                ; break if LENGTH > index

main:       jmp     main
            nop
```

A Library of Coding Primitives

Array operations that are widely used in real-life MCU applications

Usually single instruction in a high-level language: mean(array) in MATLAB
but we need to write several lines of code in assembly

or

MAD = mean(abs(array-mean(array))) in MATLAB in assembly Midterm 1

Also:

min(array) }
max(array) } **Today**

flip(array) or array.reverse()

even

multiplication }
division by a power of two } **Starting next week**

Minimum in an Array

How do we find the minimum element in an array?

One possibility is

```
min_value = array[0];
for (ii = 1; ii < length; ii++) {
    if array[ii] < min_value
        min_value = array[ii];
}
```

Can be done easily
in assembly too!

However, things get more complicated when we distinguish between different types of elements: e.g., smallest nonnegative element, smallest even number etc.

Then

```
 min_value = first even element in array
```

Not difficult
but code gets messy

Minimum in an Array

How do we find the minimum element in an array?

There is a universal initialization of min-value that results in the same exact steps of execution as previous code

```
min_value = infinity;
for (ii = 0; ii < length; ii++) {
    if array[ii] < min_value
        min_value = array[ii];
}
```

What is infinity?

The largest possible value for the type we use e.g., signed 16-bit integer, unsigned 16-bit integer etc.

$0x7FFF = 32,767_{10}$ $0xFFFF = 65,535_{10}$

With this approach we no longer need to find the first even element in the array

Maximum in an Array

How do we find the maximum element in an array?

```
max_value = -infinity;
for (ii = 0; ii < length; ii++) {
    if array[ii] > max_value
        max_value = array[ii];
}
```

where `-infinity` is the **smallest possible value** for the type we are using

e.g., signed 16-bit integer, unsigned 16-bit integer etc.

0x8000 = -32,768$_{10}$ 0

Both Min and Max in an Array

How do we find the minimum and maximum simultaneously?

and efficiently

```
min_value = infinity;
max_value = -infinity;
for (ii = 0; ii < length; ii++) {
    if array[ii] < min_value
        min_value = array[ii];
    if array[ii] > max_value
        max_value = array[ii];
}
```

Can we do any better?
i.e., reduce number of comparisons?

If length of array is n,
$\Rightarrow$ 2n comparisons

We can do 3n/2 comparisons:

Compare two elements in the array …

… compare the larger with max_value

… compare the smaller with min_value

} 3 comparisons for 2 elements

Today’s Coding Task

Task: Given an array of ten signed integers, find the min. nonnegative value

Define word `min_positive` in RAM

Create following array in FRAM

```
array: .word -37, 101, -59, -47, 23, 11, 79, -131, -5, 163
```

Easy in a high level language once we have a loop that finds the minimum

```
min_value = infinity;
for (ii = 0; ii < length; ii++) {
    if (array[ii] < min_value)  && (array[ii] >= 0)
        min_value = array[ii];
}
                                        where && = AND
```

How do we do compound conditionals in assembly?

if(cond1&&cond2)

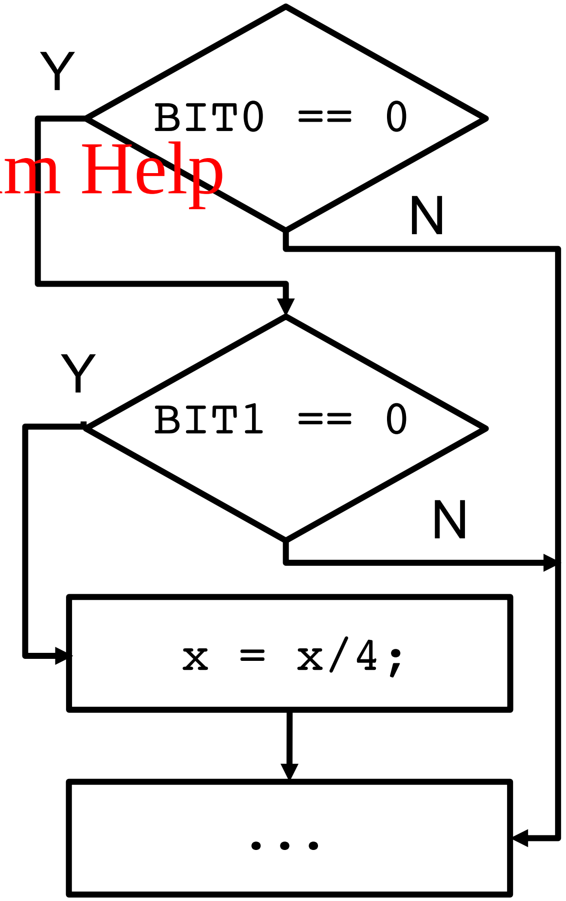
Task: If x is divisible by 4, divide it by 4

When is a number divisible by 4?

When its last two bits are both 0

```
if ( (BIT0 of x is 0)
        && (BIT1 of x is 0) )
{
     x = x/4;
}
...
```

Naïve and literal implementation



if(cond1&&cond2)

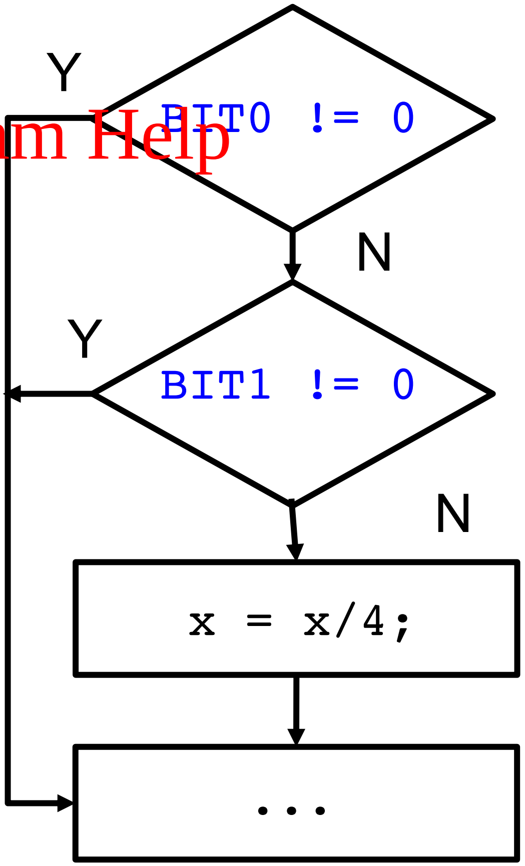
Task: If x is divisible by 4, divide it by 4

Negation to the rescue

When is a number divisible by 4?

When its last two bits are both 0

```
if ( (BIT0 of x is 0)
          && (BIT1 of x is 0) )
{
     x = x/4;
}
...
```



if(cond1||cond2)

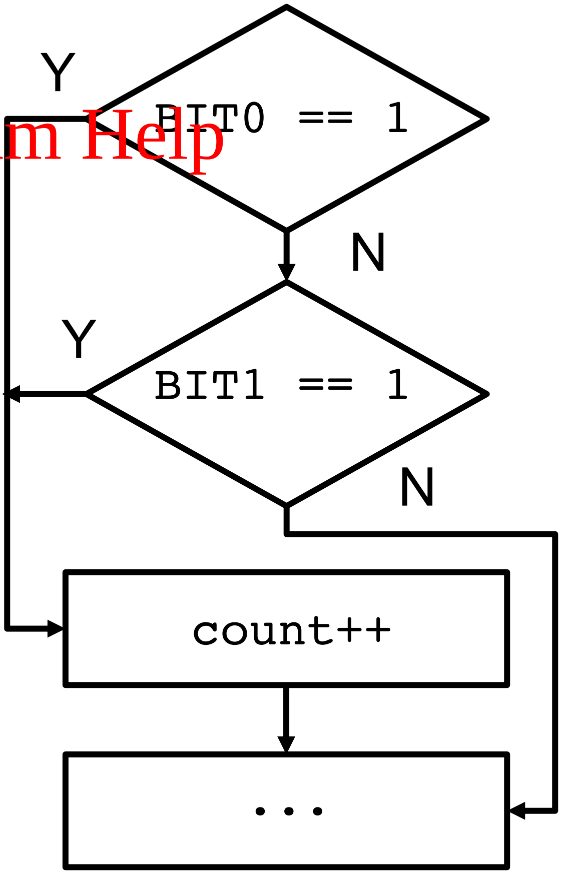
Task: Count the numbers in an array that are NOT divisible by 4

When is a number NOT divisible by 4?

When one of its last two bits are 1

```
if ( (BIT0 of x is 1)
       || (BIT1 of x is 1) )
{
       count++;
}
...
```

|| = OR

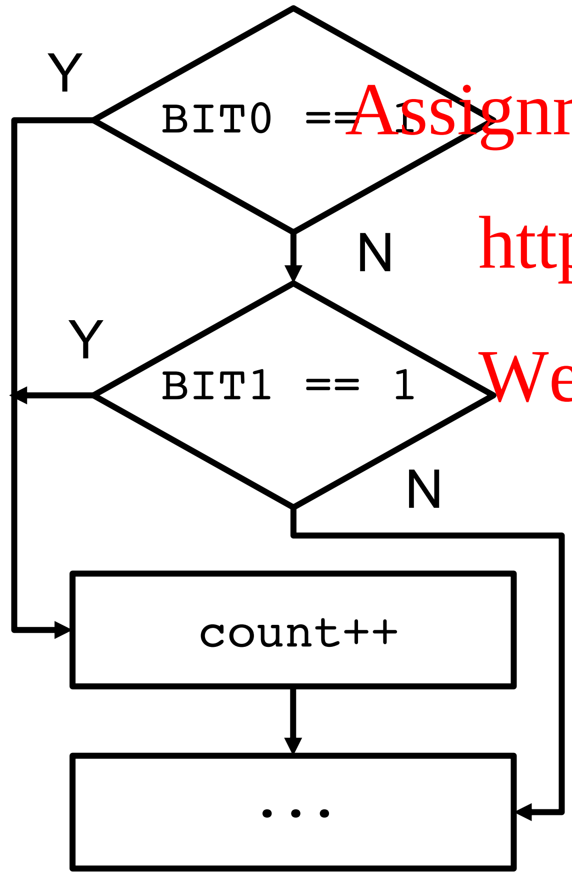


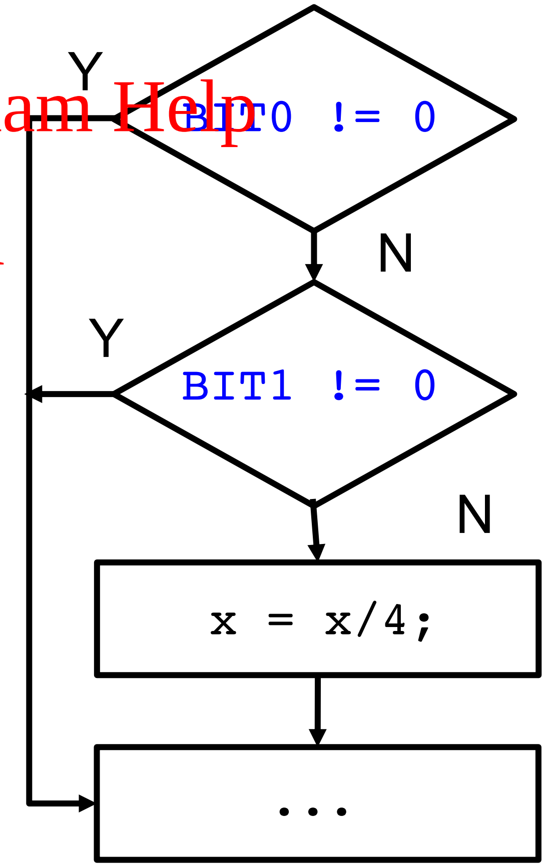
Wait a Second

In both cases we have checked the same condition: divisibility by 4

```
if (BIT0==1 || BIT1==1)
```

```
if (BIT0==0 && BIT1==0)
```





How to check divisibility by 4?

How do you check if a number is not divisible by 4?

`if (BIT0==1 || BIT1==1)` or `if ~(BIT0==0 && BIT1==0)`

Neither!

```
bit.w   #BIT1|BIT0, x          | = bitwise OR
```

The carry bit is set when either bit is set, i.e., the number is not divisible by 4

Moral of the day:

- There are no compound conditionals in assembly, use your logic
- Before starting implementing in assembly, check if you can simplify your logic
- Before solving a given problem, check if there is a simpler solution

Today’s Coding Task

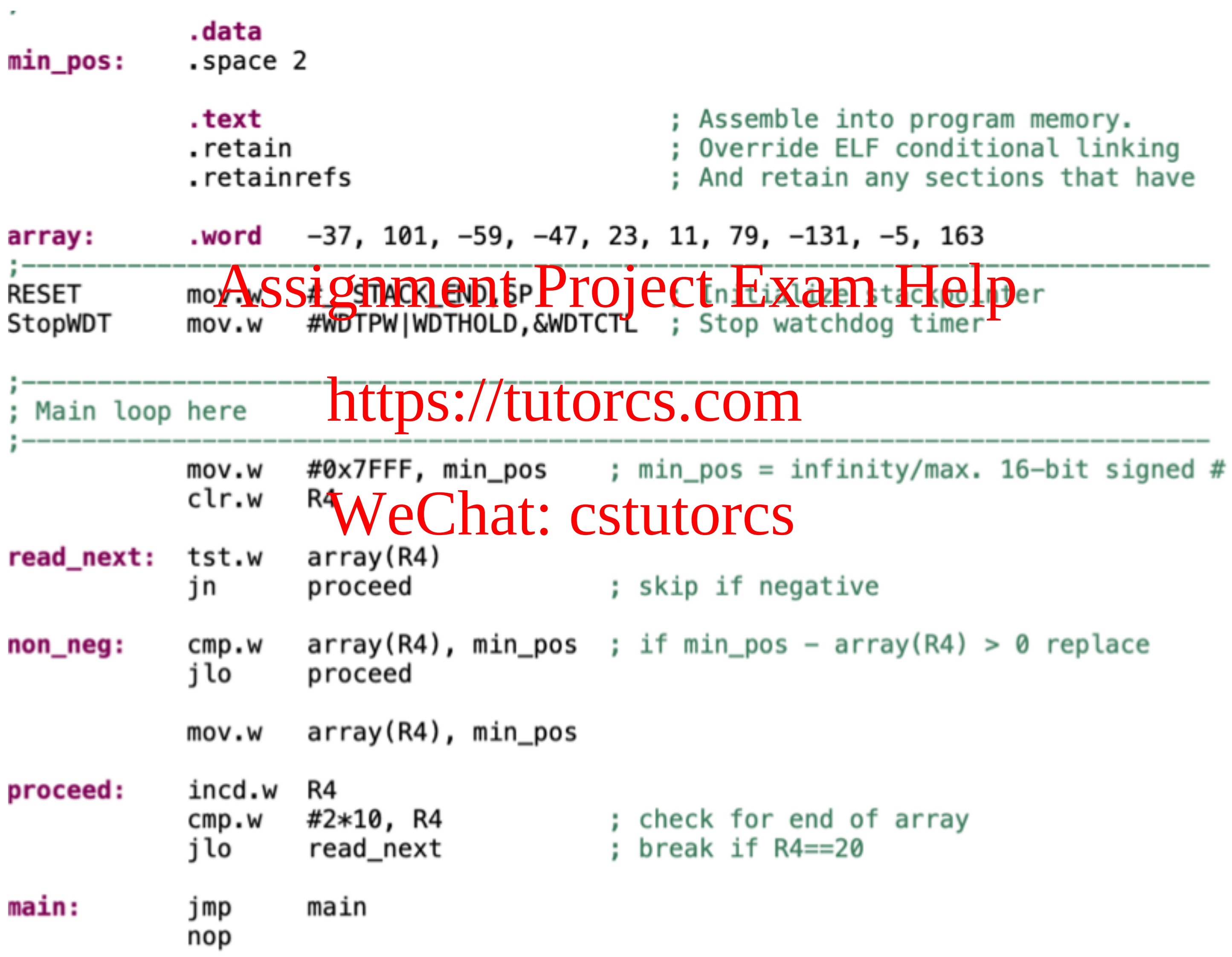
Task: Given an array of ten signed integers, find the min. nonnegative value

```
                 array+2                ...                 array+18
             array  ↓                                          ↓
array: .word -37, 101, -59, -47, 23, 11, 79, -131, -5, 163
```

How do we solve this? The questions already suggests a way:

- Loop through the array
- If current element is negative, skip to next element
- If current element is not negative, check whether it is the minimum
- Stop if done with all elements

One Solution



```
            .data
min_pos:    .space 2

            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
            .retainrefs                     ; And retain any sections that have

array:      .word   -37, 101, -59, -47, 23, 11, 79, -131, -5, 163
;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
            mov.w   #0x7FFF, min_pos     ; min_pos = infinity/max. 16-bit signed #
            clr.w   R4

read_next:  tst.w   array(R4)
            jn      proceed              ; skip if negative

non_neg:    cmp.w   array(R4), min_pos   ; if min_pos - array(R4) > 0 replace
            jlo     proceed

            mov.w   array(R4), min_pos

proceed:    incd.w  R4
            cmp.w   #2*10, R4            ; check for end of array
            jlo     read_next            ; break if R4==20

main:       jmp     main
            nop
```