another addressing mode

Indirect Register Mode (can not be used as destination)

```
mov.w  @R5, &Var
```

R5 contains an address

@R5 is the value pointed to by the address

i.e

@R5 is the value contained in the address R5

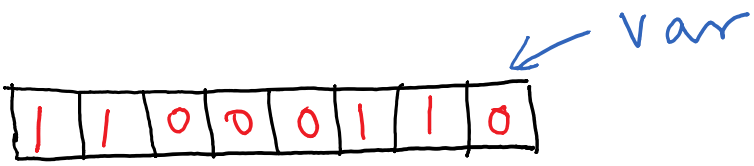
```
a:  .word  3, 5, -8, 9
```

a contains the address of the first element of a

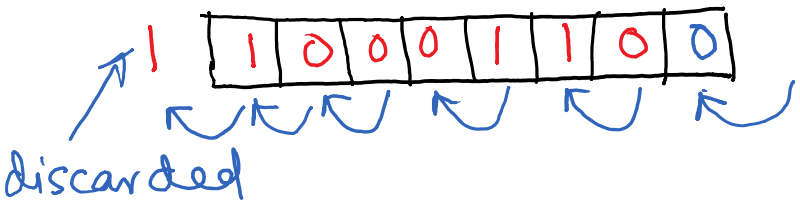
```
mov.w  #a, R5      ← now R5 contains the address of the first element of a
mov.w  @R5, &Var1  ← 3 gets copied into Var1
incd.w R5          ← move to the next element
mov.w  @R5, &Var2  ← 5 gets copied into Var2
```

Instruction rla.w (.b)

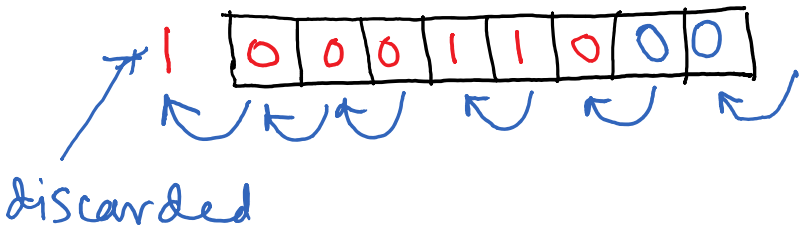
roll left arithmetic



rla.b &var

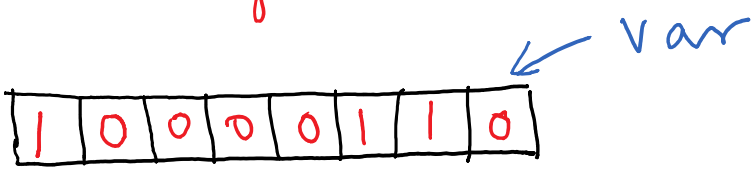


rla.b &var

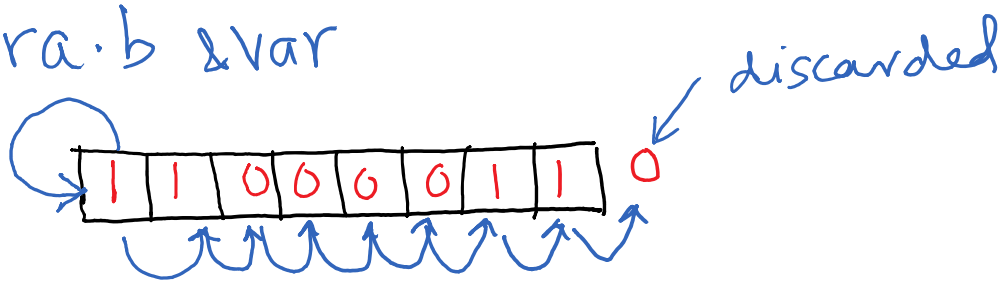


Instruction rra.w (.b)

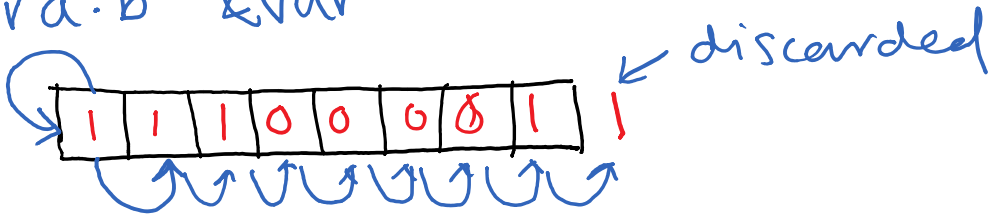
roll right arithmetic



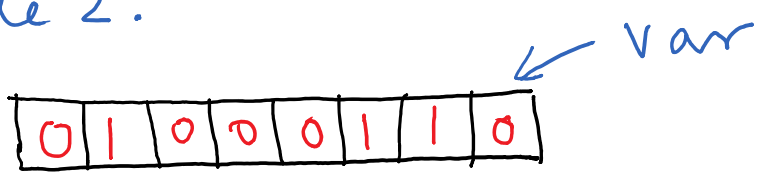
rra.b &var



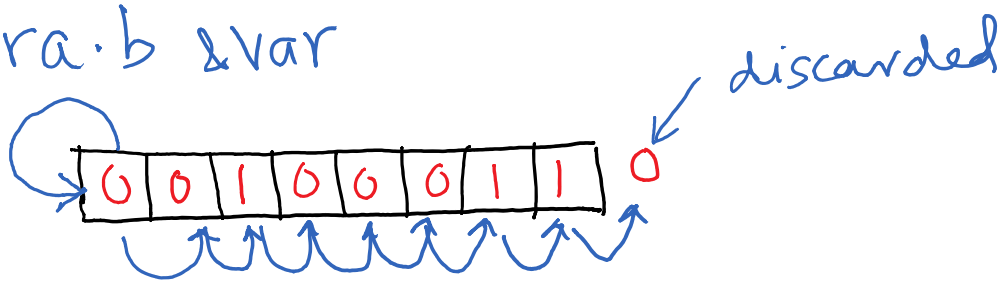
rra.b &var



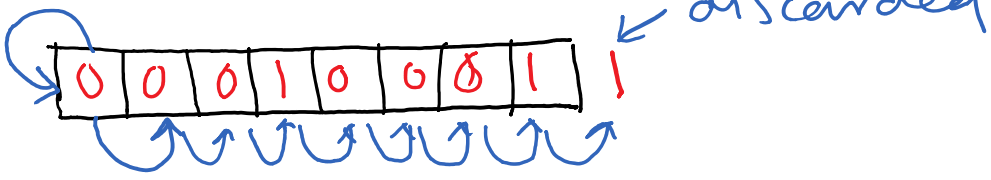
Example 2:



rra.b &var



rra.b &var



Subroutines (sub-programs)

Program: Subroutine 1

```
        .data
Result1:    .space2
Result2:    .space2
;-------------------------------------------------------------------------
        .text
                        ...
; call subroutine Times6
                mov.w   #6, R12          ; prepare input for Times6
                call    #Times6          ; call subroutine Times6
                mov.w   R13, &Result1    ; move result  to variable Result1

; call subroutine Times6
                mov     #24, R12         ; prepare input for Times6
                call    #Times6          ; call subroutine Times6
                mov.w   R13, &Result2    ;move result  to variable Result2

Loop:           jmp     Loop
;-------------------------------------------------------------------------
;                           Subroutine: Times6
;       Input: R12
;       Ouput: R13 : R12 * 6
;
;-------------------------------------------------------------------------
Times6:         mov.w   #0,   R13
                add.w   R12, R13
                add.w   R12, R13
                add.w   R12, R13
                add.w   R12, R13
                add.w   R12, R13
                add.w   R12, R13
                ret
```

Note immediate mode

execution starts from here after sub return

Call subroutine

Call subroutine again

subroutine header "contract"

"name" of subroutine

return from subroutine

```
;-----------------------------------------------------------------------
;                   Subroutine: MaxElement
;                   ----------------------
;       Find the maximum element of an array
;       Array should have word length elements
;       Input:  R10: start address of the array (R10 modified)
;       Input:  R11: length of the array in words (R11 modified)
;       Output: R12: max element of the array (R12 modified)
;       local variable R5: modified
;
;-----------------------------------------------------------------------

MaxElement:
                add.w   R11, R11        ; array elements are of word length,
                ; R11 now contains the number of bytes in the array

                mov.w   @R10, R12

                ; for-loop (alternate implementation)
                mov.w   #2, R5          ; initialize for loop
                incd    R10             ; move by one element in the array
for_cond:
                cmp.w   R11, R5
                jge     for_break       ; example (i >= 20) = ~(i < 20)
                    ; if-structure (alternate implementation)
                    cmp.w   @R10, R12
                    jge     if_break            ; (@R10 <= &max) = ~(@R10 > &max)
                    mov.w   @R10, R12
if_break:
                    ; end if-structure

                incd    R5              ; update for-loop
                incd    R10             ; move by one element in the array
                jmp     for_cond
for_break:
                ;  end for-loop
            ret
```

local register "variable"

local register "variable"

local "variable"

Calling subroutine MaxElement twice

Program : Subroutine 2

```
        .data
a1:             .word -21, 20, 30, 1, -200, -60, 0, 30, 32, 33
max1:       .space2
a2:             .word -82, 20, 30, 1, -200, -60, 0, 30, 32, 34, -90, 80
max2:       .space2
;-------------------------------------------------------------------------
        .text
            …
 ; call subroutine MaxElement
        mov.w       #a1, R10
        mov.w       #10, R11
        call        #MaxElement
        mov.w       R12, &max1

 ; call subroutine MaxElement
        mov.w       #a2, R10
        mov.w       #12, R11
        call        #MaxElement
        mov.w       R12, &max2

loop:       jmp   loop
```

Program : Subroutine 3

```
        .data
result1:        .space 2
result2:        .space 2
;-------------------------------------------------------------------------
        .text
                        …
                mov.w       #1111000001000011b, R12     ; prepare subroutine input
                call        #NoOfOnes
                mov.w       R11, &result1

                mov.w       #1111111111111111b, R12     ; prepare subroutine input
                call        #NoOfOnes
                mov.w       R11, &result2

loop:           jmp    loop

;-------------------------------------------------------------------------
;                           Subroutine: NoOfOnes
;       Counts the number of ones in the binary representation of a number
;       Input R12:    (not modified)
;       Output R11:
;       Local variable R10:  modified, not preserved
;-------------------------------------------------------------------------
NoOfOnes:
                mov.w       #0000000000000001b, R10; R10 holds bit for testing, start with the lsb
                mov.w       #0,     R11         ; R11 holds sum of set bits in the number
MoreBits:
                bit.w  R10, R12                 ; test the bit
                jnc         BitNotSet           ; if bit is not set then go to label NotSet
                inc.w  R11                      ; if you are here then bit is set, increase sum of bits
BitNotSet:                                      ; if you are here then bit is not set
                rla.w  R10                      ; move the bit to test to left by one position
                jnz         MoreBits            ; Go to label L1 if there are more bits to test
                                                ; loop is exited when R11 becomes zero and there are
                                                ; no more bits to test
                ret
```

local register "variable"

Subroutines calling subroutines

```
;-------------------------------------------------------------------------
;                       subroutine: OddFunction
;       Tests if the number of set bits in a number are odd or even
;       Input R12:  modified
;       Output R11: (R11=1 if number of set bits is odd, otherwise 0)
;       Local Variable R10: modified, not preserved
;-------------------------------------------------------------------------
OddFunction:

            call        #NoOfOnes       ; call subroutine
            mov.w    R11, R12           ; copy output of NoOfOnes to input of IsOdd
            Call        #IsOdd
            ret

;-------------------------------------------------------------------------
;                       subroutine: IsOdd
;       Tests if a number is odd or even
;       Input R12:  not modified
;       Output R11: R11 = 1 if number is odd, 0 otherwise
;-------------------------------------------------------------------------
IsOdd:
                mov.w       #0, R11
                bit.w  #0000000000000001b, R12
                jnc             Even                                    ; bit not set
                mov.w   #1, R11                                         ; bit set
Even:
                ret
```

Main program calling OddFunction

Program : Subroutine 4

```
        .data
result1:    .space 2
result2:    .space 2
;-------------------------------------------------------------------------
        .text

            mov.w   #1111000001000011b, R12   ; prepare subroutine input
            call    #OddFunction
            mov.w   R11, &result1


            mov.w   #1111111111111111b, R12   ; prepare subroutine input
            call    #OddFunction
            mov.w   R11, &result2

loop:       jmp  loop
```