In this example stack contains three words 1, 2 & 3

You can store your data in the stack. Stack grows upwards in this diagram.

SP contains 0x03FA

SP determines where the top of the stack is i.e., it determines how many words are on the stack.

Program Counter

Stack Pointer

Status Register

Constant Generator

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

General-Purpose Register

PC/R0

SP/R1

SR/CG1/R2

CG2/R3

R4

R5

R6

R7

R8

R9

R10

R11

R12

R13

R14

R15

SP contains the address (i.e., points to) of the top of the stack.
Empty Stack

RESET    mov.w  #__STACK_END, SP ; Initialize stackpointer

It is your job to initialize the stack pointer to 0x0400 at program startup
push.w 0x01111
push one word 0x1111 to the stack

SP 0x03FF

push.w 0x02222
push one word 0x2222 to the stack

SP 0x03FC
push #0x33
push one byte 0x33 to the stack
this byte remains unaltered
SP: 0x03FA

pop.b R5
pop one byte from the stack and copy it into R5
SP: 0x03FC
R5: 0x0033

Note: R5 would contain 0xCC33 if we had used pop.w R5 instead
pop W & x
pop one word from the stack and copy it into x

SP 0x03FE

x contains 0x2222

pop W & y
pop one word from the stack and copy it into y

SP 0x0400

y contains 0x1111
Program: Stack1

.data
x: .space 2
y: .space 2

; To be consistent with the class lessons, before running the program use the
; memory browser to place 0xAAAA, 0xBBBB, 0xCCCC, and 0xDDDD at the
; bottom of the stack

;-------------------------------------------------------------------------------

.text
...
push.w #0x1111
push.w #0x2222
push.b #0x33
pop.b R5 ; 0033 gets copied to R5
;pop.w R5 ; CC33 gets copied to R5
pop.w &x
pop.w &y

loop: jmp loop
Save local variables on the stack

Example: We need three word length variables count1, count2 & count3
At the beginning of your program:

```
sub.w $x6, SP ← set aside 3 words on the stack
```

```
0x0200
0x0202
```

```
RAM
```

```
0(SP) = SP → 0x03F8
2(SP) → 0x03FA
4(SP) → 0x03FC
```

Initialize variables
```
mov.w $x0, 0(SP)
mov.w $x0, 2(SP)
mov.w $x0, 4(SP)
```

Use the variables 0(SP), 2(SP) & 4(SP)

When done, always release space occupied by the variables
```
add.w $x6, SP
```

```
count3
count2
count1
```

Lesson14 Page 7
Example: Use local variables instead of global variables

.data
a: .byte 3, 6, 3, 1, 3, 6, 0, 3, 0, 3

Count how many elements of a are equal to 3, how many are equal to 6 and how many are not equal to either 3 or 6.

Pseudo-code:

count1 = 0
count2 = 0
count3 = 0

for (i = 0; i < 10; ++i)
{
    switch (a[i])
    {
        case 3:
            ++count1
            break
        case 6:
            ++count2
            break
        default:
            ++count3
            break
    }
}
Program: StackVariables1

.data
a: .byte 3, 6, 3, 1, 3, 6, 0, 3, 0, 3
;--------------------------------------------------------------
.text
...;
; create 3 word length local variables on the stack
sub.w #6, SP ; set aside space for 3 words on the stack
mov.w #0, 0(SP) ; initialize count3
mov.w #0, 2(SP) ; initialize count2
mov.w #0, 4(SP) ; initialize count1

;---------- for-loop (alternate implementation)-----------------------
mov.w #0, R5 ; i = 0
for_cond:
cmp.w #10, R5 ; (i >= 10) = ~(i<10)
jge for_break

;---------- switch-case structure --------------------------------
cmp.b #3, a(R5)
  jeq stuff1_label

  cmp.b #6, a(R5)
  jeq stuff2_label

  inc.w 0(SP) ; defaultstuff
  jmp switch_break

stuff2_label:
  inc.w 2(SP)
  jmp switch_break

stuff1_label:
  inc.w 4(SP)
  jmp switch_break

switch_break:
;---------- switch-case structure end -----------------------------
inc.w R5
jmp for_cond

for_break:
;---------- for-loop end-------------------------------------------
add.w #6, SP ; release space for 3 words from the stack

loop: jmp loop