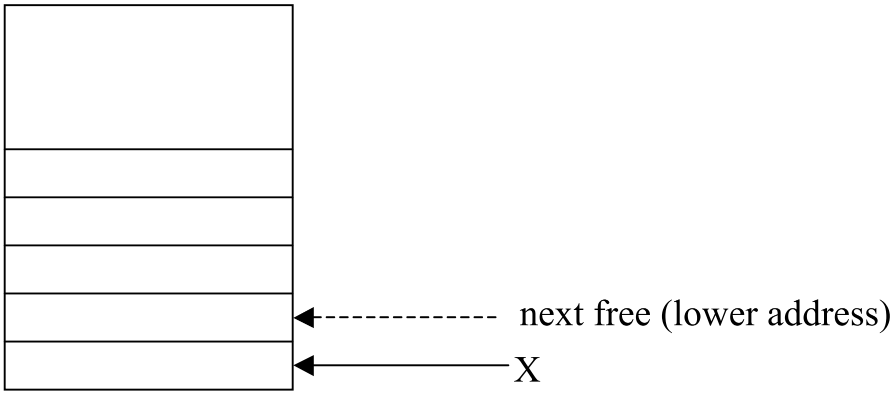
Digital and Embedded Systems
Professor Thomas Bräunl

Tutorial 3b – Stack SOLUTIONS

1. Stack

A stack data structure can use either the ATMega's system stack (using the stack pointer SP) or a user-defined stack (using index registers X, Y or Z).

Following the Atmel convention of **post-decrement** for **push** and **pre-increment** for **pop**, implement your own subroutines *mypush* and *mypop*. Use X as a user stack-pointer and R16 as contents to be pushed/poped.



Solution:

Note1: Ideally, we would want to use the comands STS X-, R16 for push and LDS R16, +X for pop. Unfortunately, these pre/post operations are not available (only -X and X+), so we have to program them step-by-step.

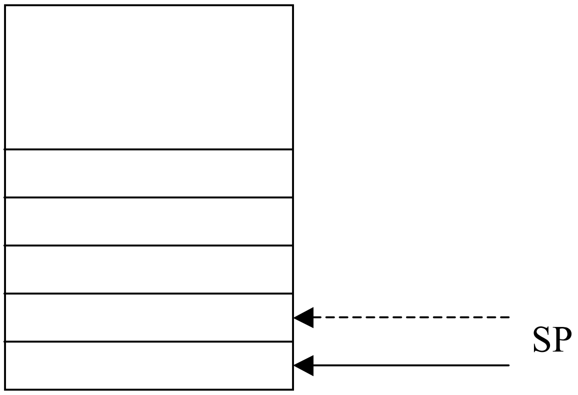
Note2: There are no operations like INC X or DEC X available. Those only work for R0..R31 (8-bit registers).

```
mypush: ST X, R16
        SBIW XH:XL, 1; dec. X
        RET

mypop: ADIW XH:XL, 1; inc. X
       LD   R16, X
       RET
```

2. PUSH and POP

Execute the following Atmel code, draw the stack contents, and find out the register contents at the end. Assume SP is initialized with $04FF.

```
LDI  R16, $10
LDI  R17, $20

PUSH R16
PUSH R17
NOP
POP  R16
POP  R17
```



Contents of: R16 _~~$10~~; $20___________________

R17 _~~$20~~; $10___________________

SP _~~$04FF~~; ~~$04FE~~; ~~$04FD~~; ~~$04FE~~; $04FF______________

3. Machine Code and Status Register

Consider the machine program shown below. Fill in the instruction and parameters column by finding each instruction in the instruction set.

Address	Code	Instruction	Parameters
0	2700	CLR	R16
1	2711	CLR	R17
2	0000	NOP	
3	9100	LDS	R16, $0400
4	0400		
5	9110	LDS	R17, $0401
6	0401		
7	0F01	ADD	R16, R17
8	9300	STS	$0402, R16
9	0402		

Fill in the blanks in the program execution table below. Each line corresponds to a single instruction.

Enter the contents of the program counter, registers, memory locations and status register flags after each instruction is executed. Status flags N, Z, V and C stand for the negative, zero, overflow and carry flag, respectively.

Note:

- *Assume all flags are initially set to zero.*
- *The PC only increments by 1 or 2, depending whether an instruction takes 1 or 2 words (2 or 4 bytes).*

(before ex)

PC	R16	R17	$0400	$0401	$0402	V	N	Z	C
0	$4A	$7E	$55	$AC	$42	0	0	0	0
1	$0	$7E	$55	$AC	$42	0	0	1	0
2	$0	0	$55	$AC	$42	0	0	1	0
3	$0	0	$55	$AC	$42	0	0	1	0
5	$55	0	$55	$AC	$42	0	0	1	0
7	$55	$AC	$55	$AC	$42	0	0	1	0
8	$01	$AC	$55	$AC	$42	0	0	0	1
A	$01	$AC	$55	$AC	$01	0	0	0	1

Note:

- *The zero flag is initially a 1 due to the CLR operations. It does not change in the NOP instruction.*
- *The zero flag changes to a 0 in the ADD instruction because the result of the addition was 01 which is not equal to 0*
- *The carry flag is set in the ADD operation as the total result would have been $101.*