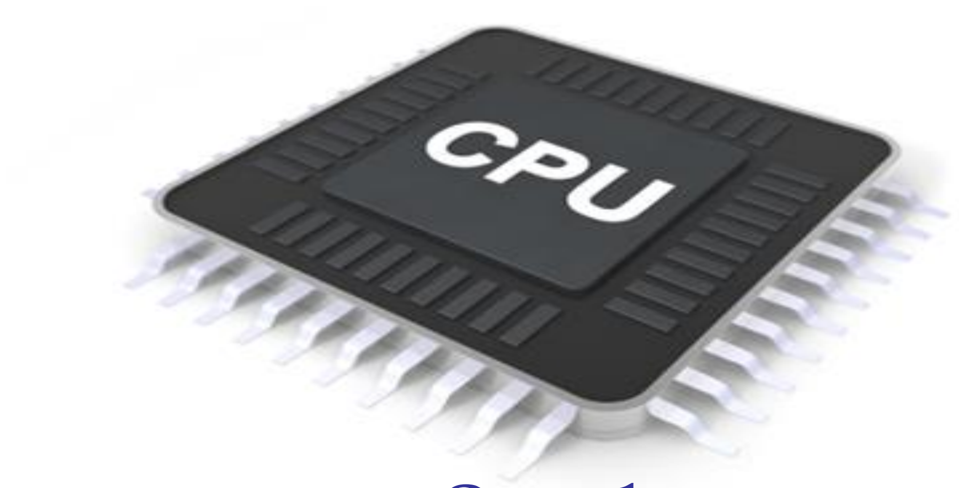

Stack

ENEE 3582

Microp

Run-time Stack

- Area in memory (RAM)
  - Defined at the end of the data RAM
- Used for temporary storage
  - by programmer
    - to store temporary values instead if creating variables
    - This is how local variables are created inside functions.
  - by specific instructions
    - CALL, RET: used stack to store return address of the function call
- Has a different and unique functions for storage and retrieval
  - PUSH is used to store into the stack
  - POP is used to retrieve from the stack

Stack Pointer (SP)

- SP is a 16-bit IO register used to point to current top of the stack
  - IO registers use IN and OUT to move data to/from them
  - 16 bit register: SPH, and SPL
- At the beginning SP points to the RAMEND
  - Data stores in the stack are created starting at RAMEND
  - MEGA 2560:
    - Size = 8KB
    - RAM start is `0x0200`
    - RAMEND is address `0x21FF`

Changing/Copying SP

- To re-set SP to RAM-end:

```
LDI R16, HIGH(RAMEND)    ;load SPH
OUT SPH, R16             ;
LDI R16, LOW(RAMEND)     ;load SPL
OUT SPL, R16
```

  - Theoretically, RAMEND can be replaced by any number < `0x21FF`
  - Careful: Changing SP will cause losing the top of the stack

- To copy SP into another register:

```
IN XH, SPH               ;X = SP
IN XL, SPL               ;
```

PUSH

- Creates data on the top of the stack
- Data is added to RAMEND, going backward towards RAM start
- SP is decremented by the number of bytes PUSHed into the stack
  - PUSH operates on registers => 1 byte pushed (decrements by 1)
  - Data is stored
  - SP is decremented (post-decrement)
- Format: `PUSH Rm              ;Mem[SP-] = Rm`

POP

- Removes data from the top of the stack
- Last in, first out
- Data is removed, going forward towards RAMEND
- SP is incremented by the number of bytes POPed from the stack
  - PUSH operates on registers => 1 byte POP (increment by 1)
  - SP is incremented (pre-increment)
  - Then the data is removed first
- Format: `POP Rm                 ;Rm = Mem[+SP]`

LIFO

- Stack is a temporary storaege
  - Everything pushed into the stack must be removed
- Last in First Out
- When using PUSH and POP: POP in reverse order to PUSH

Coding Example 1/2

```
LDI R16, 16         ;0x10
LDI R17, 17         ;0x11
LDI R18, 18         ;0x12
LDI R19, 19         ;0x13
                    ;SP=0x21FF
                    ;0x21F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PUSH R16            ;SP=0x21FE
                    ;0x21F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10
PUSH R17            ;SP=0x21FD
                    ;0x21F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 11 10
PUSH R18            ;SP=0x21FC
                    ;0x21F0  00 00 00 00 00 00 00 00 00 00 00 00 00 12 11 10
PUSH R19            ;SP=0x21FB
                    ;0x21F0  00 00 00 00 00 00 00 00 00 00 00 00 13 12 11 10
```

Coding Example 2/2

```
PUSH R16        ;mem[0x21FF] = 0x10
PUSH R17        ;mem[0x21FE] = 0x11
PUSH R18        ;mem[0x21FD] = 0x12
PUSH R19        ;mem[0x21FC] = 0x13
                ;0x21F0  00 00 00 00 00 00 00 00 00 00 00 13 12 11 10
                ;SP= 0x21FB
POP R16         ;SP= 0x21FC  , R16=0x13
POP R17         ;SP= 0x21FD  , R17=0x12
POP R18         ;SP= 0x21FE  , R18=0x11
POP R19         ;SP= 0x21FF  , R19=0x10
```

Coding Exercise

- `Xarr` is stored in the data memory. Write a program to reverse the order of array values in `Xarr` without using the stack and using the stack. Assume the length of the array is stored in `len`.