MC

Unterprogramme

- selbstständige Programme
- vom Hauptprogramm aufgerufen
- Hauptprogramm unterbrochen
- beliebig oft von versch. Stellen aufrufbar

Stack

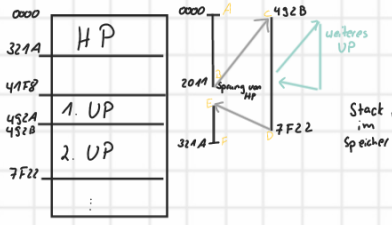
- Ende des Speichers
- Merker für Programm rückkehr
- am Ende des Speichers
- Adresse beginnt bei FFFF
- arbeitet Rückwärts

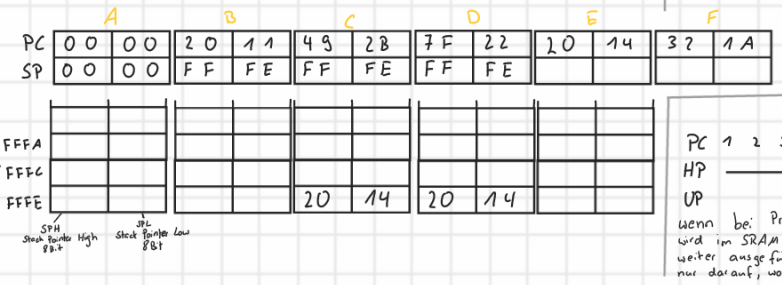
Stack Pointer (SP)

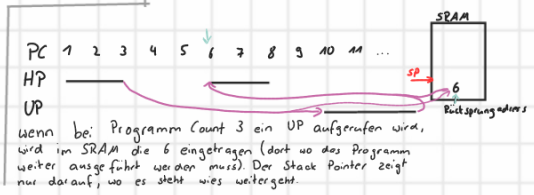
- Zeiger / Verweis
- zeigt auf den Bereich in dem die Rücksprungadresse steht

Programm Counter (PC)

- sagt Stelle an dem Programm läuft







Wenn bei Programm Count 3 ein UP aufgerufen wird, wird im SRAM die 6 eingetragen (dort wo das Programm weiter ausgeführt werden muss). Der Stack Pointer zeigt nur darauf, wo es steht wies weitergeht.

Interrupts

- auf bestimmtes Ereignis reagieren
- von Hardware ausgelöste Unterbrechung des Programmablaufs
- einzelne Hardware können interrupt auslösen
- Bsp: Tätigkeit einer Sekretärin
  - ↳ versch. Tätigkeiten nacheinander
  - ↳ unterschiedliche Prioritäten
  - ↳ Anruf Chef ↑Prio Kaffee trinken ↓Prio
- bei µC sind Tätigkeiten die Hardware
  - ↳ z.B. AD-Wandler, Timer, externe Interrupts, RESET
- je nach Prio der Interrupts wird entschieden
  - ↳ INT0 → INT1 → INT2
- quasi parallel zum Hauptprogramm
  - ↳ da nur eine CPU → keine echte Parallelität
- bei Interrupt → Hauptprogramm unterbrochen
  - ↳ Interruptroutine ausgeführt
  - ↳ zurück zum Hauptprogramm

Polling

Signale regelmäßig in while Schleife abfragen und Signale auswerten

Nachteil: -nicht sofort reagiert
-mit Verzögerung

Bei kritischen Signalen z.B. Not-Aus nicht geeignet

Interrupt

wird sofort reagiert wenn Interrupt Signal auftritt wird Hauptprogramm unterbrochen

Vorteil: + kein Zeitverzug

Bedingungen:

```
# include <avr/io.h>
# include <avr/interrupt.h>
ISR (INT0_vect) {
  // Interrupt Service Routine → hier die weiteren Befehle der ISR
  // während abarbeiten der ISR wird I-Flag des SREG zurückgesetzt
     ↳ Folgen: andere ISR sind gesperrt / können nicht ausgelöst werden
  // erst nach ISR Ende wird I-Flag zurückgesetzt → andere Interrupts entsperrt
  // wenn das nicht gewollt muss I-Flag per Befehl sei() gesetzt werden
}

void main (void) {
    // Port konfigurieren → Verhalten wenn ISR ausgelöst wird
    // hier beim loslassen "rising edge"
    // INT0 konfg: MCUCR (MCU controll Reg)
    MCUCR = (1<<ISC01) | (1 << ISC00);
    // hier wird Interrupt INT0 gesetzt
    // INT0 freigabe: GICR (General Int Cont Reg)
    GICR = (1 << INT0);
    // allge. Interrupt-Freigabe durch I-Flag
    sei();
    while (1) { // Hauptprogramm
    } // Ende while
} // Ende main
```