Overzicht Programmeren In Assembler Voor De 8051

Algemeen

- waardes worden voorafgegaan door een spoorwegteken (#) en afgesloten met een indicatie van het talstelsel (b=binair, d=decimaal, h=hexadecimaal)
  hexadecimale waardes die beginnen met een letter krijgen een 0 na het spoorwegteken
- commentaar wordt voorafgegaan door een dubbele slash (//) of een puntkomma (;)
- het dollarteken ($) representeert de huidige regel
  $-1 is de voorgaande regel, $-2 de regel daarvoor, $+1 is de volgende regel, enz.

Instructies

Bitinstructies		
clr	clear	zet bit op 0
setb	set bit	zet bit op 1
cpl	complement	bitwaarde complementeren
Spronginstructies		
jmp	jump	onvoorwaardelijk: ga naar de opgegeven locatie
djnz	decrement and jump if not zero	voorwaardelijk: verlaag de waarde met 1 en ga naar de opgegeven locatie indien de waarde niet gelijk is aan 0
jb	jump if bit set	voorwaardelijk: ga naar de opgegeven locatie indien de bit gelijk is aan 1 [jb bit,locatie]
jnb	jump if not bit set	voorwaardelijk: ga naar de opgegeven locatie indien de bit niet gelijk is aan 1 [jnb bit,locatie]
jc	jump if carry set	voorwaardelijk: ga naar de opgegeven locatie indien de carry gelijk is aan 1
jnc	jump if not carry set	voorwaardelijk: ga naar de opgegeven locatie indien de carry gelijk is aan 0
cjne	compare and jump if not equal	voorwaardelijk: vergelijk de twee eerste argumenten, indien ze verschillen, ga naar de opgegeven locatie
Andere instructies		
mov	move	kopieer opgegeven waarde naar adres [mov adres,waarde]
dec	decrement	verlaag de waarde met 1
inc	increment	verhoog de waarde met 1
rl	rotate left	roteer accumulator naar links, bit 7 gaat naar bit 0 [rl A]
rlc	rotate left through carry	roteer accumulator naar links, bit 7 gaat naar carry, carry gaat naar bit 0 [rlc A]
rr	rotate right	roteer accumulator naar rechts, bit 0 gaat naar bit 7 [rr A]
rrc	rotate right through carry	roteer accumulator naar rechts, bit 0 gaat naar carry, carry gaat naar bit 7 [rrc A]
add	add accumulator	verhoog accumulator met opgegeven waarde [add A,waarde]
addc	add accumulator with carry	verhoog accumulator met opgegeven waarde en carry bit [addc A,waarde]
subb	subtract from accumulator with borrow	verminder accumulator met opgegeven waarde [subb A,waarde]
mul	multiply accumulator with borrow	vermenigvuldig accumulator (A) met B, LSB komt in A, MSB komt in B [mul AB]
div	divide accumulator by borrow	deel accumulator (A) door B, quotiënt komt in A, rest in B [div AB]
push	push	plaats waarde van opgegeven adres (!) op de stack, wordt intern voorafgegaan door verhoging van stack pointer
pop	pop	haalt bovenste waarde van stack en plaatst die in het opgegeven adres, wordt intern gevolgd door een verlaging van stack pointer

Adresnamen

- R0-R7 = 8 registers
  de registers zijn bitadresseerbaar
  R0-1 zijn als pointers bruikbaar [@R0 en @R1]
- A = accumulator
- B = borrow
- C = carry bit, is 1 als de waarde in de accumulator groter is dan 255
- SP = stack pointer

Interessante pagina's in de handleiding

057 – ADC0CN: ADC0 control register
066 – ADC0 karakteristieken
108 – REF0CON: reference control register & voltage reference electrical characteristics
→ internal temperature sensor on/off
→ 00000011b bij gebruik van ADC !
→ Vref = 2.43 V
147 – locatie van de interrupts
→ 000Bh = timer0 overflow
→ 0023h = UART0 received/transmitted
149 – IE: interrupt enable
→ ES0: UART0 interrupt on/off
→ ET0: timer0 interrupt on/of
→ EX0: external interrupt on/off (enablen indien computer waarde opvraagt !)
217 – Priority crossbar decode table
224 – XBR0: port I/O crossbar register 0
→ UART0 on/off
→ UART0: transmit TX via P0.0
receive RX via P0.1
227 – P0 data register & P0MDOUT output mode register
264 – UART0 operational modes
265 – UART0 baudrate-formule
271 – SCON0: UART0 control register
→ mode instellen
→ TI0: transmission interrupt on/off
→ RI0: receive interrupt on/off
(wordt ook opgeworpen als je zelf iets in SBUF plaatst om het te verzenden !)
272 – SSTA0: UART0 status and clock selection register
→ baudrate delen door twee on/off
289 – TCON: timer control register
→ TR: timer run control
→ TF: timer overflow vlag
290 – TMOD: timer mode register
→ T0M0 & T0M1
291 – CKCON: clock control register
→ T0M
292 – TL0 & TH0: timer low byte & timer high byte

Programmastructuren

1) main programma

```
$include (c8051f120.inc)
cseg at 0000h                 ➔ locatie main-module
jmp main
cseg at 0050h                 ➔ schrijf onderstaande code in het programmageheugen
main:                                 (te beginnen bij adres 0050h)
        clr EA                ➔ disable alle interrupts
        mov WDTCN,#0DEh       ➔ WatchDog Timer CoNtrol register: watchdog uitschakelen
        mov WDTCN,#0ADh               (zorgt anders voor reset bij oneindige lus)
        setb EA               ➔ enable alle interrupts
        mov SFRPAGE,#0Fh      ➔ locatie van XBR2 (Port I/O Crossbar register 2)
        mov XBR2,#40h         ➔ aanzetten van crossbar (anders P0-3 geblokkeerd)
        ... main code ...     ➔ bevat altijd oneindige lus (jmp $)
end                           ➔ 1x, helemaal op einde
```

2) subroutine (in main code)

```
…
call naam                     ➔ aanroepen van subroutine
…
naam:
        … actie …
ret                           ➔ einde van subroutine
```

3) interrupt routine (ISR = Interrupt Search Routine)

```
$include (c8051f120.inc)
cseg at 0000h
jmp main
cseg at 000Bh          ➔ locatie timer-interrupt
jmp ISRTR0             ➔ Interrupt Search Routine TimeR
cseg at 0023h          ➔ locatie UART-interrupt
jmp ISRUART0           ➔ Interrupt Search Routine UART (seriële poort)
cseg at 0050h
main:
        clr EA
        mov WDTCN,#0DEh
        mov WDTCN,#0ADh
        setb EA
        mov SFRPAGE,#0Fh
        mov XBR2,#40h
        ... main code ...
        ISRTR0:
                ... interrupt code ...
        reti                          ➔ einde van ISR
        ISRUART0:
                … interrupt code …
        reti
end
```

- modules die aangeroepen worden in ISR ook beëindigen met reti !
- binnen ISR kunnen externe interrupts uit- en ingeschakeld worden met clr EX0 en setb EX0
- interruptvlaggen worden niet automatisch gewist !

4) dubbele vertragingslus: totale vertraging = 256 x 256 klokcycli

```
lus1:
        … afwisselende actie …
        mov R1,#0FFh            ➔ R1 = 256
        lus2:
                mov R0,#0FFh    ➔ R0 = 256
                djnz R0,$       ➔ R0 aftellen naar 0
                djnz R1,lus2    ➔ R1 aftellen naar 0, indien != nul terug R0 aftellen
                jmp lus1        ➔ terug naar actie
```

5) poort: P0

```
mov P0MDOUT,#00001111b   ➔ P0.0-0.3 = input, P0.4-0.7 = output
```

- P4-7 worden opgeslaan in SFRPAGE #0Fh !

6) timer: TR0: interval van 1 sec

```
mov SFRPAGE,#00h        ➔ timerinstellingen gebeuren in SFRPAGE 0 !
mov TMOD,#01h           ➔ timer0 mode 1 (16 bit timer)
mov CKCON,#02h          ➔ SYSCLK delen door 48
mov TH0,#06h            ➔ telregisters instellen: TH0 (high byte) & TL0 (low byte)
mov TL0,#0C5h                 (alleen startwaarde aanpasbaar, loopt altijd tot FFFF)
setb TR0                ➔ timer0 starten
loop:
        jnb TF0,$       ➔ wacht 1 sec → timer0 overflow
        clr TF0         ➔ timer0 overflow vlag wissen
        … actie …
        clr TR0         ➔ timer0 stoppen
        mov TH0,#06h    ➔ telregisters opnieuw instellen
        mov TL0,#0C5h
        setb TR0        ➔ timer0 opnieuw starten
        jmp loop
```

- startwaarde en instelling SYSCLK berekenen:
  - klokfrequentie = 24.5 MHz = 24.5E6 Hz (intern gedeeld door 8)
    duur 1 timertick = 1 / klokfrequentie
  - tijdsinterval = 1s = # ticks / klokfrequentie
    → # ticks in 1s = 1 s * 24.5E6 Hz / 8 / 48 = 63802.08 = 63802
    → omdat waarde groter is dan 65536 is extra deling van ingangskloksignaal nodig
    (keuze uit 8, 12, 48 → keuze bepaald door uitkomst: zo klein mogelijke rest na komma)
  - startwaarde = 65535 - 63802 = 1733 = 06C5

7) klavier input

- 1 toets

```
mov P0MDOUT,#00010000b   ➔ P0.4 = output
clr P0.4                 ➔ P0.4 = 0
wacht:
    jb P0.0,$            ➔ wacht tot toets ingedrukt wordt
    … actie …
    jnb P0.0,$           ➔ wacht tot toets gelost wordt
    jmp wacht
```

- 16 toetsen: 4 groepjes van 6 regels (1 clear, 1 set, 4 controles)
  P0.7 instellen op 1 / P0.4 instellen op 0 / P0.0-3 overlopen
  P0.4 instellen op 1 / P0.5 instellen op 0 / P0.0-3 overlopen
  P0.5 instellen op 1 / P0.6 instellen op 0 / P0.0-3 overlopen
  P0.6 instellen op 1 / P0.7 instellen op 0 / P0.0-3 overlopen

8) stack: 2 x 3 = ?

```
mov A,#02d
mov B,#03d

push Acc            ➔ push gebeurt met adressen, Acc = adres van accumulator
push B
call multiply
pop 00h             ➔ haal A en B van stack (wegschrijven naar R0)
pop 00h
jmp $

multiply:
    push 00h        ➔ push R0 op stack (registers R0→R7 = adressen 00h→07h)
    mov R0,SP       ➔ maak kopie van stackpointer
    dec R0          ➔ verlaag R0 tot plaats van eerste data (*)
    dec R0
    dec R0
    mov B,@R0       ➔ kopieer data waar R0 naar wijst in B
    dec R0
    mov A,@R0       ➔ kopieer data waar R0 naar wijst in A
    mul AB          ➔ LSB van A x B wordt in A geplaatst (MSB in B) → A = 2 x 3 = 6
    pop 00h         ➔ verwijder R0 van stack
ret
```

- (*) hou er rekening mee dat ook het returnadres van de subroutine (2 bytes) op de stack gepusht worden bij het aanroepen van een subroutine
  → zo kan de configuratie van voor de subroutineaanroep hersteld worden na het beëindigen van de subroutine
  → figuurtje maken !
- geheugenlay-out van de 8051: zie p27

9) display output: teller (van 0 tot en met 9 en herbeginnen)

```
mov R0,#20h              ➔ bitpatronen opslaan in geheugenadressen vanaf adres 20h
                           (makkelijk aanpasbaar indien meer geheugenadressen nodig)
mov @R0,#00111111b       ➔ 0, gebruik van R0 als pointer
inc R0
mov @R0,#00000110b       ➔ 1
inc R0
mov @R0,#01011011b       ➔ 2
inc R0
mov @R0,#01001111b       ➔ 3
inc R0
mov @R0,#01100110b       ➔ 4
inc R0
mov @R0,#01101101b       ➔ 5
inc R0
mov @R0,#01111101b       ➔ 6
inc R0
mov @R0,#00000111b       ➔ 7
inc R0
mov @R0,#01111111b       ➔ 8
inc R0
mov @R0,#01101111b       ➔ 9
mov R0,#20h              ➔ index terug op eerste adres instellen

mov P0,#00h              ➔ alle LED's uit
mov R2,#00h              ➔ index = 0
jmp uitschrijven

test:
        jb P3.7,$        ➔ wacht tot drukknop ingedrukt
        jnb P3.7,$       ➔ wacht tot drukknop gelost
        inc R2           ➔ index incrementeren
        cjne R2,#10d,uitschrijven ➔ R2 gelijk aan 10 ? nee → uitschrijven, ja → ga verder
        mov R2,#00d
        jmp uitschrijven

uitschrijven:
        mov A,R0         ➔ waarde van R1 berekenen = startadres (R0) + index (R2)
        add A,R2
        mov R1,A
        mov P0,@R1       ➔ gebruik van R1 als pointer
        jmp test
```

10) interne temperatuursensor + ADC

```
setb AD0EN              ➔ enable ADC0 (ADC = Analoog Digitaal Converter)
clr AD0TM               ➔ ADC0 tracking mode 0
clr AD0CM0              ➔ volgende twee bits clearen om conversie te kunnen starten
clr AD0CM1                 met ADOBUSY=1
mov AMX0SL,#08h         ➔ ADC0 multiplexer channel select
mov REF0CN,#07h         ➔ temperatuursensor inschakelen
                              + zetten van nodige bits voor gebruik ADC
conversie:
        clr AD0INT      ➔ disable interrupts
        setb AD0BUSY    ➔ start conversie
        jnb AD0INT,$    ➔ pollen: wacht tot klaar met omzetten (geeft interrupt)
        … actie …       ➔ verwerk ADC0L (low byte) & ADC0H (high byte)
        jmp conversie          (ADC0 is slechts 12-bit ADC !
```

- ADC0 is een 12-bit ADC
  → de eerste 4 bits van ADC0H vormen een tekenextensie en bevatten geen meetwaarde
- interpretatie van ADC0H en ADC0L:
  vb: ADC0H = 05 & ADC0L = 1B
  → ADC0 = 0000 | 0101 0001 1011
  = (0/2 + 1/4 + 0/8 + 1/16 + 0/32 + 0/64 + 0/128 + 1/256 +
  1/512 + 0/1024 + 1/2048 + 1/4096) * Vref
  = 0,38459 * 2.43 V
  = 0,93456 V
- berekening fysische temperatuur
  → referentie: 0.00286 * 15°C + 0.776 = 0.8189 V (formule zie grafiek p50)
  = 0000 | 0101 0110 0100 (ADC0H & ADC0L) (*)
  → nauwkeurigheid van temperatuursensor (=0.2°C) kan je zelf narekenen of aflezen op p66
  → plaats referentiewaarde (*) in geheugen en verhoog telkens met 1 tot huidige waarde van ADC0 bereikt is, elke verhoging staat voor een temperatuurstijging van 0.2°C t.o.v. 15°C
- clr AD0TM (is niet noodzakelijk voor het programma, 0 is de reset value van AD0TM)
  → ADC controleert continue, tenzij er een interrupt bezig is

11) seriële poort: UART0

```
mov XBR0,#04h           ➔ enable UART0
mov P0MDOUT,#01h        ➔ UART0 transmit = P0.0, UART0 receive = P0.1
mov SFRPAGE,#00h        ➔ UART instellingen gebeuren in SFRPAGE 0 !
mov SCON0,#01000000b    ➔ UART0 mode 1 (= 8-bit, variabele baudrate)
mov SSTA0,#00h          ➔ SMOD0 = 0

mov TMOD,#20h           ➔ timer1 mode
mov CKCON,#00010000b    ➔ timer1 gebruikt SYSCLK (T1M = 1)
mov TH1,#246d           ➔ startwaarde timer (**)
setb TR1                ➔ timer1 starten

lus:
        mov A,#0FFh
        mov SBUF0,A     ➔ accumulator op buffer van serieel kanaal plaatsen
        jnb TI0,$       ➔ wachten bewerking voltooid (transmission interrupt)
        jmp lus
```

- (**) startwaarde van timer1 berekend met formule voor baudrate p265
  → hier: stel baudrate = 9600 en laat vervolgens SMOD0 en T1M variëren
  → kies de configuratie waarvoor de rest na de komma van TH1 minimaal is