TI-Übung 2

6809-Assembler

Andreas I. Schmied
(andreas.schmied@uni-ulm.de)

AspectIX-Team
Abteilung Verteilte Systeme
Universität Ulm

WS2005

6809 – Wiederholung (1)

- 8/16-Bit Prozessor
- Register: A/B/D, S, U ,X, Y, CC, DP, PC
- Adressierung: Immediate, Direct, Indexed, Indirect, Autoinc/dec
- Nicht alle Adressierungsmodi/Register kombinierbar!

6809 – Wiederholung (2)

- Condition Codes
  - Inhalt: EFHINZVC
  - Manipulation, z.B. Carry setzen

```
1  set   orcc  #%00000001
2        orcc  #$01
3  clr   andcc #%11111110
4        andcc #$FE
```

  - Änderungen durch Operationen beachten
    - siehe Datenblatt

6809 – Assembler a09 (1)

- Symbole
  - Label = „Data at Address“ (Alias für Adressnr.)
  - #Label = „Address of Label“
- Explizite Sprunggrößen
  - Short Jump: JMP <$0
  - Long Jump: JMP >$0
  - PC-relative: BRA *
- Listing Files „.l“ beachten
- Datenblätter nutzen!

6809 – Assembler a09 (2)

- Aufbau unserer Übungsquellcodes

```
1  org   $400
2        jmp start
3  data  fcc ”text”
4        fcb 0
5        fdb $1234
6  dend
7  start
8        ldu #stack  ; init stack pointer
9        ...
10       swi         ; return to monitor
11
12 org   $700        ; assert this to be
13 stackend          ; after code region!
14       rmb $100
15 stack             ; grows downward
```


Add32Bit – Aufgabe

- Addiere 32bit-Zahlen
  - $44ff3311 und $54014422
- Liegen im Speicher bei Labels e und f
- Ergebnis soll in e abgelegt werden
- Beispiel-Datei add.a

Add32Bit – Implementierung

```
1  ; e   is $44ff3311
2  ; f   is $54014422
3  ; e+f is $99007733
4
5          org $400
6          ldx #e       ; x $\to$ e
7          ldy #f       ; y $\to$ f
8          ldb #3       ; add bytes 0..3
9
10         andcc #$fe
11 next
12         lda   b,x
13         adca  b,y
14         sta   b,x
15         decb
16         bpl   next
17
18         swi
19
20 e       fcb $44, $ff, $33, $11 ; $99007733
21 f       fcb $54, $01, $44, $22
```


CRC – Allgemeines

- Prüfsummenberechung
- Aufspüren von Übertragungsfehlern
- Basiert auf Polynomdivison in $\mathbb{Z}_2$
- Beispiel-Datei crc.a

Literatur

- The CRC Pitstop: `http://www.ross.net/crc/`
  - Ross N. Williams
  - siehe Paper „A painless guide to CRC...“
- Englische Wikipedia: CRC32
- CRC-Calculator + Links
  `http://www.zorc.breitbandkatze.de/crc.html`

CRC – Algorithmus

- Variante „SIMPLE“
  - Prüfpolynom ist x^8 x^2 x^1 +1 → „Poly“=0x07
  - Rest-Register := 0
  - 8 Bits an Daten anhängen
  - Solange Datenbits vorhanden
    - Schiebe Rest 1 Bit nach links
    - Fülle von rechts mit Datenbit
    - Bei Carry Rest := Rest XOR Poly
  - CRC-8-Ergebnis ist im Rest-Register
- Realisierung

  A – Rest und Ergebnis
  B – Aktuelles Datenbyte
  X – Index in Datenstring
  Y – Bitzähler des aktuellen Bytes
  U – Index in Register-Log

CRC – Rahmen

```
1         org $400
2         jmp start
3
4   data  fcc "TI2 Uebung2"    ; input data
5   stuff fcb 0                ; stuff bits for division end
6   dend
7   loga  rmb (dend-data)*8    ; for reg A values
8   logb  rmb (dend-data)*8    ; for reg B values
9
10  poly  fcb $07              ; crc-8 generator polynom
11
12  start
13        ...
14  done
15        swi                  ; crc remainder is now in reg a
```

CRC – Implementierung

```
1  start   clra                  ; clear remainder
2          ldx  #0               ; set index to first data byte
3          ldu  #loga            ; set pointer to register log
4  loopchr
5          ldb  data,x           ; load a character
6          ldy  #8               ; character has 8 bits
7  loopbit
8          stb  (dend-data)*8,u  ; log b
9          sta  ,u+              ; log a and inc pointer
10
11         lslb                  ; next data bit into carry (c:=MSB)
12         rola                  ; roll in this carry
13         bcc  nextbit          ; no div when no remainder-carry
14         eora poly             ; div mod 2 by poly
15 nextbit
16         leay -1,y             ; decrement y = "one bit less left"
17         bne  loopbit          ; goto leftshift of next bit if y!=0
18 nextchr
19         leax 1,x              ; increment x = "next data byte"
20         cmpx #(dend-data)     ; if X=datalength then done
21         bne  loopchr          ; else load new character
22 done
```


Stack – Allgemeines (1)

- Unterprogrammaufruf, lokale Variablen, ...
- Monitor/JSR/BSR nutzen Register S, Userstack U
  - S mit $400 initialisiert
  - ➜ „xl400“ lädt Binaries direkt über Systemstack
- Initialisieren und Benutzen
  - Register zeigen auf „Top of Stack“
  - Benötigte Stackgröße abschätzen
  - Unterlaufgrenze beachten!

Stack – Allgemeines (2)

- Push
  - Registersatz: PSHU r,...
  - Einzelwerte: STr ,−U bzw. ,−−U
- Pull
  - Registersatz: PULU r,...
  - Einzelwerte: LDr ,U+ bzw. ,U++
- Pull/Push verändern Flags nicht
  - Restauration der Register vor RTS
- Registersätze in festgelegter Reihenfolge

Stack – Allgemeines (3)

- JSR vs. BSR
  - JSR bietet Adressierungsmodi
  - Subroutine, Rekursion, ...
- Tracing im Monitor
  - t für Single-Step, p für Step-Over
- Warum sind STr ,U− und LDr,+U nicht implementiert?
  - ➜ Wegen Stackaufbau: U/S zeigen auf „Top of Stack“

String-Funktionen

- Null-terminierte Strings
- Stringlänge berechnen
- Substring-Muster suchen

- Realisierung
  - Übergabe von Parametern
  - Rückgabe von Ergebnis/Status
  - Register veränderbar?
  - Register oder Stack nutzen?
- Beispiel-Datei str.a

String-Funktionen – Rahmen (unvollständig)

```
1         org $400
2         jmp start
3
4   string  fcc ”cacao”
5   stringz fcb 0
6   search  fcc ”cao”
7   searchz fcb 0
8
9   start
10
11        ldx #string
12        jsr strlen    ; strlen(”cacao”)
13
14        ldx #string
15        ldy #search
16        jsr substr    ; substr(”cacao”, ”cao”)
17        swi
18        ...
```

String-Funktionen – Rahmen (inkl. Stack)

```
1         org $400
2         jmp start
3
4   string  fcc ”cacao”
5   stringz fcb 0
6   search  fcc ”cao”
7   searchz fcb 0
8
9   start   ldu #stack
10
11        ldx #string
12        jsr strlen    ; strlen(”cacao”)
13
14        ldx #string
15        ldy #search
16        jsr substr    ; substr(”cacao”, ”cao”)
17        swi
18        ...
19        org $700
20  stend   rmb $100
21  stack
```


String-Funktionen – Strlen

- Stringlänge berechnen
- Null suchen, Mitzählen

- Realisierung
  - Param X = Adresse des Strings
  - Return A = Länge

String-Funktionen – Implementierung strlen

```
1  strlen
2          clra
3          pshu b,x
4  slLoop
5          ldb  ,x+
6          beq  slEnd
7          inca
8          bra  slLoop
9  slEnd
10         pulu b,x
11         rts
```


String-Funktionen – Substr

- Substring-Muster suchen
  - Mustervergleich Zeichenweise
  - Bei Fehler
    - Muster zurücksetzen
    - ➜ Sonderfall?
    - Wenn Fehler bei 1. Stringzeichen
    - Dann nächstes Stringzeichen prüfen
  - Sonst nächstes Zeichen
  - Muster komplett gefunden
  - Stringende erreicht → Musterende erreicht?
- Realisierung
  - Param X = Text String
  - Param Y = Query Substring
  - Return X = Substring Pointer
  - When Z=1

String-Funktionen – Implementierung substr

```
1  substr  pshu a,y
2  ssReset
3          stx  -2,u
4  ssLoop
5          tst  ,y
6          beq  ssEnd
7          lda  ,x+
8          beq  ssEnd
9          cmpa ,y+
10         beq  ssLoop
11 ssMiss
12         ldy  1,u      ; reset y (see push layout)
13         leax -1,x     ; undo x+
14
15         cmpx -2,u     ; if failed at first char
16         bne  ssReset
17         leax 1,x      ; then advance to next
18         bra  ssReset
19 ssEnd
20         ldx  -2,u
21         tst  ,y
22         pulu a,y      ; keeps CC
23         rts
```