Experimentelle Hardwareprojekte
Lehrstuhl für Advanced Computing
Institut für Informatik

Versuch A4

Assemblerprogrammierung eines RISC-Prozessors

Online-Version

Andreas Reinsch

Inhaltsverzeichnis

Der Versuch führt am Beispiel des DLXJ-Prozessors [2], einer sehr vereinfachten Variante des RISC-Prozessors DLX [1] in die Assemblerprogrammierung und in das Hardware-Debugging ein. Durch entsprechende Erweiterungen gestattet der Prozessor die Beobachtung aller Bussignale, aller Registerinhalte und der Speicherinhalte. Die Befehlsabarbeitung kann unterbrochen werden:

- nach jedem Takt,
- nach einer festlegbaren Anzahl von Takten,
- nachdem ein Befehl vollständig abgearbeitet wurde und
- an einem Unterbrechungspunkt (Codezeilennummer).

1. Versuchsvorbereitung

1. Informieren Sie sich über die Möglichkeiten des Hardware-Debuggings des DLXJ-Prozessorsystems im Abschnitt 5.4 in [2].

2. Die Berechnung der Fibonaccischen Zahlen erfolgt nach folgender Rekursionsformel:

$$a_n = a_{(n-1)} + a_{(n-2)} \quad \text{für} \quad n = 3, 4, 5, \ldots \\ \text{Startwerte: } a_1 = 0,\ a_2 = 1; \tag{1}$$

   Entwickeln Sie zwei Programme, so daß jeweils die vollständige Zahlenfolge entsprechend der Gleichung (1) für $n = 3 \ldots 12$ generiert wird:

   a) mit dem nichterweiterten Befehlssatz (Tabelle 1, Abschnitt 2 in [2]).

   b) mit dem erweiterten Befehlssatz (Tabelle 1, Abschnitt 2 und Tabelle 3, Abschnitt 4.2 in [2]) .

   Beide Programme sollen nacheinander ausgeführt werden. Die Ergebnisse sollen im RAM des Prozessors ab der Adresse 0x1000_0000 gespeichert werden, beginnend mit den Ergebnissen des Programms mit erweitertem Befehlssatz. Kommentieren Sie jede Ihrer Anweisungen. Ein Programmbeispiel finden Sie im Anhang B in [2]. Durch eine entsprechende Anweisung am Ende des Programms ist zu verhindern, daß der Prozessor abstürzen kann (die Anweisung „end“ im Programm im Anhang B in [2] ist lediglich eine Assembleranweisung, zusätzlicher Programmcode wird damit nicht generiert).

3. Die Ergebnisse stehen im Dualcode im Speicher und werden vom Debugger im Hexadezimalcode angezeigt. Im Interesse der besseren Lesbarkeit ist es sinnvoll, eine Dual-BCD-Umsetzung durchzuführen. Informieren Sie sich über geeignete Algorithmen zur Dual-BCD-Umsetzung. Das Prinzip und ein mögliches Verfahren sind im Anhang A ausführlich beschrieben.

4. Im Anhang B finden Sie ein Assemblerprogramm, mit dem die Dual-BCD-Umsetzung realisiert werden kann. Analysieren Sie das Programm und kommentieren Sie jede Anweisung ausführlich.

1.1. Vorbereitung des Arbeitsplatzes

Die Vorgehensweise ist analog zum Versuch A3. Die versuchsspezifischen entsprechenden Befehle lauten `a4_COPY` und `a4`. Den Zugang zum Server und eine X Window Umgebung benötigen Sie für diesen Versuch nur, wenn Sie Ihre Ergebnisse (.pdf-Datei) bereits am Arbeitsplatzrechner betrachten möchten.

Die Abbildung 1 zeigt alle im Verlauf des vorherigen Versuchs A3 verwendeten Programme. Für den Versuch A4 benötigen Sie die Scripte `dlxj_configure_fpga`, `dlxj_program_rom` und `dlxj_debug_processor`.

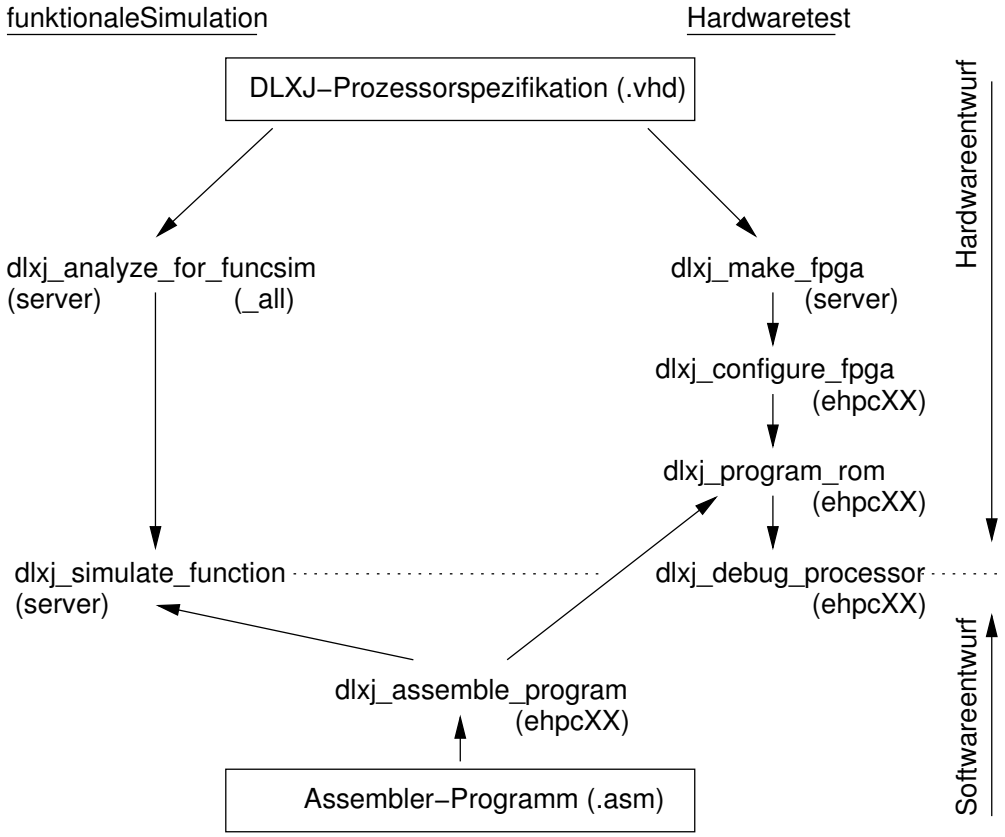

Abbildung 1: Der Entwurfsfluß

Für die weitere Arbeit beachten Sie bitte folgendes:

Für alle Dateien, die im Verlauf des Versuchs eventuell modifiziert oder erstellt werden müssen, soll das Verzeichnis ∼`/dlxj/asm` (Assemblerprogramme) verwendet werden. Arbeiten Sie grundsätzlich nicht außerhalb des Verzeichnisbaums ∼/dlxj. Der Befehl `dlxj_editor` ruft den GUI-basierten Editor `gedit` auf. Bei geringer Datenrate des Netzwerks ist es günstiger, mit einem Editor ohne GUI zu arbeiten. Geeignet sind z.B. `nano` oder `vi`.

2. Versuchsdurchführung

2.1. Implementierung und Test der Assemblerprogramme

Beide in der Versuchsvorbereitung entworfenen Programme für die Berechnung der Fibonaccireihe sollen nun gemeinsam in einer Assemblerdatei editiert und getestet werden. In einer zweiten Assemblerdatei soll ein Programm erstellt werden, welches unter Verwendung Ihres Programms zur Berechnung der Fibonaccireihe mit erweitertem Befehlscode und unter Verwendung des Programms nach Anhang B
(Datei:`~/dlxj/asm/dual_bcd.asm`) die Ergebnisse sowohl im Dualcode als auch im BCD-Code erzeugt und ab der Adresse 0x1000_0000 im RAM abpeichert. Die Reihenfolge der Berechnung und der Speicherung der Ergebnisse wählen Sie bitte selbst (z.B.: können zuerst alle Zahlen erzeugt werden, so daß das bereits vorhandene Programm weiterverwendet werden kann und dann erfolgt die Dual-BCD-Umwandlung).

Hinweise für die Assemblerbenutzung : Immediate-Werte können dezimal (z.B.: 21), binär (z.B.: B"10101"), hexadezimal (z.B.: X"15"), oder oktal (z.B.: O"25") angegeben werden. Anstelle eines Immediate-Wertes kann ein statischer Ausdruck mit den Operationen $+$, $-$, $*$ stehen. Der Assembler erwartet als letztes Zeichen „Newline“ - (Enter); Sonderzeichen sind generell nicht zugelassen. Der erste Befehl darf nicht mit einer Marke beginnen (evtl. einen NOP-Befehl einfügen).

Bei jeder Programmentwicklung müssen folgende Arbeitsschritte ausgeführt werden:

1. Editieren Sie Ihre Assemblerdatei. Als Vorlage kann das Programm `~/dlxj/asm/inc2_dec1.asm` genutzt werden.
2. Assemblieren Sie die Datei und achten Sie dabei auf eventuelle Fehlerausgaben.
3. Konfigurieren Sie das FPGA (`dlxj_configure_fpga`).
4. Übertragen Sie den Maschinencode in den Programmspeicher des DLXJ-Prozessors (`dlxj_program_rom`).
5. Starten Sie den Debugger (`dlxj_debug_processor`) und testen Sie Ihr Assemblerprogramm. Wählen Sie dazu selbständig geeignete Debuggerbefehle aus.
6. Beobachten Sie das Verhalten Ihres Programms. Korrigieren Sie eventuelle Fehler (Korrektur - Assemblerlauf - Programmspeicher aktualisieren). Protokollieren Sie den Testverlauf.
7. Drucken Sie das Ergebnis aus, nachdem das Programm vollständig abgearbeitet wurde (Debuggerbefehl: `srr`, `debug_results.pdf` im Verzeichnis `prn`).
8. Drucken Sie das funktionsfähige, gut kommentierte Assemblerprogramm aus.

3. Versuchsauswertung

Mit den im Verlauf des Versuchs entstandenen Aufzeichnungen, Änderungen an den VHDL-Dateien (nur die Änderungen in das Protokoll aufnehmen) und Ausdrucken ist der Gliederung der Versuchsdurchführung entsprechend ein Protokoll anzufertigen. Die Versuchsvorbereitung ist zusammmen mit dem Protokoll abzugeben.

Anhang A Dual-BCD-Umsetzung

Bei der Dual-BCD-Umsetzung können Pseudotetraden $(0d10 \ldots 0d15)$ auftreten. Eine Korrekturmöglichkeit besteht darin, die Dualzahl von rechts nach links in einen BCD-Rahmen[1] zu schieben. Man beginnt mit dem höchstwertigen Bit der Dualzahl. Wenn eine 1 eine Tetradengrenze überschreitet, dann entsteht ein Fehler, der durch Addition von 6 zu der niederwertigeren Tetrade korrigiert wird. Treten nach dem Schieben Pseudotetraden auf (unabhängig von einer Grenzüberschreitung einer 1), so sind diese ebenfalls durch Addition von 6 in der betreffenden Tetrade mit Übertrag von 1 in die nächsthöhere Tetrade zu korrigieren. Erst dann darf erneut geschoben werden.

Eine weitere Korrekturmöglichkeit besteht darin, immer dann vor dem Schieben 3 zu addieren, wenn der Wert einer Tetrade > 4 ist. Hier sind die folgenden beiden Situationen möglich:

Wenn der Wert < 8 ist tritt keine Grenzüberschreitung auf (das höchste Tetradenbit ist ja 0). Beim Schieben entsteht aber eine Pseudotetrade, die man vor dem Schieben durch Addition von 3 zu dieser Tetrade korrigiert. Ist der Wert ≥ 8, so wird durch das Schieben die Tetradengrenze von einer 1 überschritten. Auch hier muß durch Addition von 3 korrigiert werden. Aufgrund der Korrektur können höhere Tetradenwerte als 9 nicht auftreten.

Das folgende Java-Programm veranschaulicht das beschriebene Verfahren für die Dual-BCD-Umsetzung einer Dualzahl in eine maximal vierstellige Dezimalzahl:

```
public class dualbcd
{    public static void main(String[ ] args)
    {   int bcd=0, dual=0xc6, i=0;
        for(i=7;i>-1;i--)
        {       System.out.format("i=%2d  ", i);
                System.out.format("Tetr.1: %02x   ",bcd & 0xf);
                if((bcd &  0xf)    >    0x4) {bcd += 0x3;}
                System.out.format("korr: %02x   ",bcd & 0xf);
                System.out.format("Tetr.2: %02x   ",bcd & 0xf0);
                if((bcd &  0xf0)   >   0x40) {bcd += 0x30;}
                System.out.format("korr: %02x   ",bcd & 0xf0);
                bcd = bcd<<1;
                bcd = ((dual>>i) & 0x1) | bcd;
                System.out.format("SLL, Bit anfügen: %3x %n",bcd);
        }
        System.out.format("Dualzahl: %4x BCD-Zahl: %4x %n",dual,bcd);
    }
}
```

[1] Bei einem BCD-Rahmen repräsentiert eine Tetrade eine Dezimalstelle, d.h jeder Dezimalstelle (Einer, Zehner, Hunderter ...) ist einer Tetrade zugeordnet.

Das Programm produziert folgende Ausgaben:

```
i= 7 Tetr.1: 00 korr: 00 Tetr.2: 00 korr: 00 SLL, Bit anfügen:   1
i= 6 Tetr.1: 01 korr: 01 Tetr.2: 00 korr: 00 SLL, Bit anfügen:   3
i= 5 Tetr.1: 03 korr: 03 Tetr.2: 00 korr: 00 SLL, Bit anfügen:   6
i= 4 Tetr.1: 06 korr: 09 Tetr.2: 00 korr: 00 SLL, Bit anfügen:  12
i= 3 Tetr.1: 02 korr: 02 Tetr.2: 10 korr: 10 SLL, Bit anfügen:  24
i= 2 Tetr.1: 04 korr: 04 Tetr.2: 20 korr: 20 SLL, Bit anfügen:  49
i= 1 Tetr.1: 09 korr: 0c Tetr.2: 40 korr: 40 SLL, Bit anfügen:  99
i= 0 Tetr.1: 09 korr: 0c Tetr.2: 90 korr: c0 SLL, Bit anfügen: 198
 Dualzahl:   c6  BCD-Zahl:  198
```

Anhang B Programm für die Dual-BCD-Umsetzung

```
;-----------------------------------------------------;
; Datei         : dual_bcd.asm                        ;
; Datum         : Mar. 2012                           ;
; Beschreibung  : Dual-BCD-Umsetzung                  ;
;-----------------------------------------------------;
;
        org   X"00000000"  ; Programmstartadresse
        start init         ; markiert den ersten Befehl
; Immediate-Werte fuer Speicheroperationen:
        zero  equ  X"0000" ; Offset 0
        disp  equ  X"0FFC" ; Adressierung des letzten
                           ; Wortes im ROM (4KByte)
        ram   equ  X"0FF8" ; Adressierung des vorletzten
                           ; Wortes im ROM
; Die letzten beiden ROM-Speicherzellen sind mit zwei
; Konstanten initialisiert. In der Speicherzelle mit der
; Adresse 0x0000_0FFC steht die Adresse fuer das
; Display Latch und in der Speicherzelle 0x0000_0FF8
; wurde die Basisadresse des RAM abgelegt.
;
; Die in eine BCD-Zahl zu wandelnde Dualzahl sei:
dual_high equ X"0"
dual_low  equ X"1e"
;
; In Binaerdarstellung 1 Tetrade und 1 Bit:
dualbits   equ 5
;
; Die Darstellung als BCD-Zahl erfordert 2 Ziffern:
digits     equ 2
;
;  verwendete Register:
;  --------------------
; r1  Dualzahl
; r2  aktueller BCD-Wert
; r3  initiale Tetradenmaske (X"f")
; r4  aktuelle Tetradenmaske
; r5  initialer Tetradenvergleichswert (4+1)
; r6  aktueller Tetradenvergleichswert
; r7  initialer Tetradenkorrekturwert (3)
; r8  aktueller Tetradenkorrekturwert
; r9  Laufvariable i (Dualzahlindex)
; r10 Tetradenindex
```

```
; r11 Sprungbedingung fuer Tetradenkorrektur
; r12 Abbruchbedingung fuer die Tetradenkorrektur
; r13 aktueller Dualwert
; r14 aktuelles Dualzahlbit
; r31 Speicheradresse des Display-Latches
;
;; Initialisierung:
;; ---------------
init:
        nop
ini_bcd or.i r1,r0,dual_high
        sll.i r1,r1,16
        or.i r1,r1,dual_low
        add.i r9,r0, dualbits
        add.i r2,r0,0
        add.i r3,r0,X"f"
        add.i r5,r0,5
        add.i r7,r0,3
        lw.i r31,disp(r0)
;; BCD-Korrektur:
;; -------------
bcd     sub.i r9,r9,1
        add.i r10,r0,0-4
        srl r13,r1,r9
        and.i r14,r13,1
        sll.i r2,r2,1
        or r2,r14,r2
        beqz r9,displ
te_ko   add.i r10,r10,4
        sub.i r12,r10,(digits-1) * 4
        sll r4,r3,r10
        sll r6,r5,r10
        sll r8,r7,r10
        and r14,r2,r4
        slt r11,r14,r6
        beqz r11,plus_3
        beqz r12,bcd
        j te_ko
plus_3  add r2,r2,r8
        beqz r12,bcd
        j te_ko
displ   sw.i zero(r31),r2
ende    j ende
end
```

Literatur

[1] Hennessy, J.L. and Patterson, D.A. *Rechnerarchitektur – Analyse, Entwurf Implementierung, Bewertung*. Friedr. Vieweg Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994.

[2] A. Reinsch. *Der DLXJ RISC-Prozessor (Skript zur Lehrveranstaltung Experimentelle Hardwareprojekte)*. FSU Jena, Institut für Informatik, 2020.