

HÖHERE TECHNISCHE BUNDESLEHRANSTALT

HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Fach: HWE	Hausübung – Nr.: 2	Datum der Abgabe: 12.11.2015
Lehrer: Dipl. -Ing. Hermann Dum	Benennung: Assembler Befehle in RTL	
Klasse / Jahrgang: 5BHEL	Name: Lukas Graussam	Kat.-Nr. 07

Inhaltsverzeichnis

1	Aufgabenstellung	3
2	Arithmetischer Befehl	3
2.1	Beschreibung	3
2.2	Mikrooperationen	3
2.3	VHDL Test-Bench Code	4
2.4	Simulation	5
3	Transfer-Befehl	5
3.1	Beschreibung	5
3.2	Mikrooperationen	6
3.3	Simulation	6
3.4	VHDL Test-Bench Code	7
4	Sprung-Befehl	8
4.1	Beschreibung	8
4.2	Mikrooperationen	8
4.3	Simulation	8
4.4	VHDL Test-Bench Code	9

1 Aufgabenstellung

Aufgabe war es, sich 3 Assembler Befehle auszudenken, wobei ein Arithmetischer, ein Transfer und ein Sprung / Call / Branch Befehl enthalten sein sollte. Die Befehle sollten dann mit Worten beschrieben werden, und im RTL Format dargestellt werden. Des Weiteren sollten die einzelnen Mikrooperationen der Befehle in RTL + Steuersignale dargelegt werden. Abschließend sollten die gesamten Befehle mit dem Xilinx Webpack in einer Von-Neumann Maschine simuliert werden. Diese wurde uns zur Verfügung gestellt, sodass nur mehr die Simulationen getan werden mussten.

2 Arithmetischer Befehl

2.1 Beschreibung

Gewählter Befehl: SUBINBR x

Beispiel: SUBINBR #0x2

Dieser Befehl soll eine Subtraktion durchführen, wobei der Input-Wert von dem Immediate Wert x abgezogen werden soll (Immediate Wert steht im Memory nach dem OP-Code). Das Ergebnis diese Subtraktion soll in das BR-Register gespeichert werden.

RTL Beschreibung: $BR \leftarrow x - I_{port}$

2.2 Mikrooperationen

1: PCe, IRw, MEMe:	$IR \leftarrow MEM[PC]$
2: INe, ADs, alufun, ARw:	$AKKU \leftarrow Input_Port$
3: PCe, ADs, ALUe, alufun, PCw:	$PC \leftarrow PC + 1$
4: PCe, MEMe, ADs, alufun, ARw:	$AKKU \leftarrow MEM[PC] - AKKU$
5: ARw, ALUe, alufun, BRw:	$BR \leftarrow AKKU$
6: PCe, ADs, ALUe, alufun, PCw:	$PC \leftarrow PC + 1$

2.3 VHDL Test-Bench Code

Hier der relevante Code der Testbench:

```
stim_p: process
Begin

IPort<="00000010";  -- Input Wert

wait for 103 ns; -- 1. Takt

MEMw<='0';MAw<='0';MAe<='0';MEMe<='0';ALUe<='0';ADs<='0';ARw<='0';alufun<=
"0000";PCw<='0';PCe<='0';PCde<='0';SPe<='0';INe<='0';OUTw<='0';IRw<='0';SPw
<='0';BRw<='0';BRe<='0'; -- Alle auf Null setzen

PCe<='1'; IRw<='1'; MEMe<='1';  --  IR <- MEM[PC]

wait for 100 ns; -- 2. Takt
-- Alle auf Null setzen

INe<='1'; ADs<='0'; alufun<="0110"; ARw<='1';  --  AKKU <- IN

wait for 100 ns; -- 3. Takt
-- Alle auf Null setzen

PCe<='1'; ADs<='1'; ALUe<='1'; alufun<="1010"; PCw<='1';  --  PC <- PC +1

wait for 100 ns; -- 4. Takt
-- Alle auf Null setzen

PCe<='1'; MEMe<='1'; ADs<='0'; alufun<="0101"; ARw<='1';
--  AKKU <- MEM[PC] - AKKU

wait for 100 ns; -- 5. Takt
-- Alle auf Null setzen

ARw<='1'; ALUe<='1'; alufun<="0000"; BRw<='1';  --  BR <- AKKU

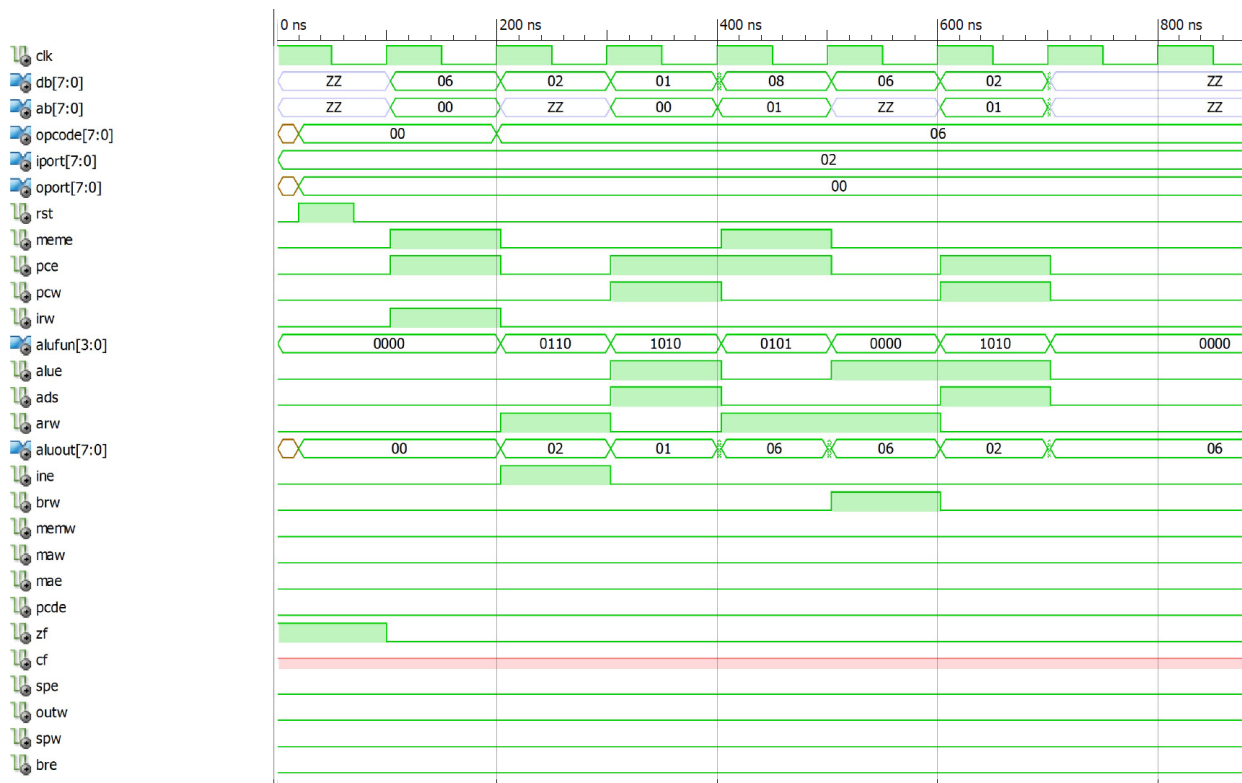
wait for 100 ns; -- 6. Takt
-- Alle auf Null setzen

PCe<='1'; ADs<='1'; ALUe<='1'; alufun<="1010"; PCw<='1';  --  PC <- PC + 1

wait for 100 ns; -- 7. Takt
-- Alle auf Null setzen
-- Letzte Takte für schöne Simulation normalerweise noch hier
End process;
```

Anmerkung: Nach jedem Takt wurde wieder jedes Signal wie beim 1. Takt auf null gesetzt. Auf Grund der Länge wurde dies hier, sowie bei den folgenden Programmcodes durch Kommentare ersetzt.

2.4 Simulation



Hier die erfolgreiche Simulation.

In diesem Fall ist der Immediate Wert 0x8 und der Input Wert 0x2 (siehe Code). Im vorletzten Takt kann man erkennen, dass die Subtraktion funktioniert, da über den Datenbus der richtige Wert 0x6 in das BR-Register geschrieben wird.

3 Transfer-Befehl

3.1 Beschreibung

Gewählter Befehl: INDMOV x

Beispiel: INDMOV #0x2

Es handelt sich bei diesem Befehl um eine indizierte Adressierung. Der Inhalt der Speicheradresse, welche aus der Summe des BR-Register-Werts und dem Operanden x besteht, soll in den Stack geschrieben werden. Der Operand x steht im Memory nach dem OP-Code.

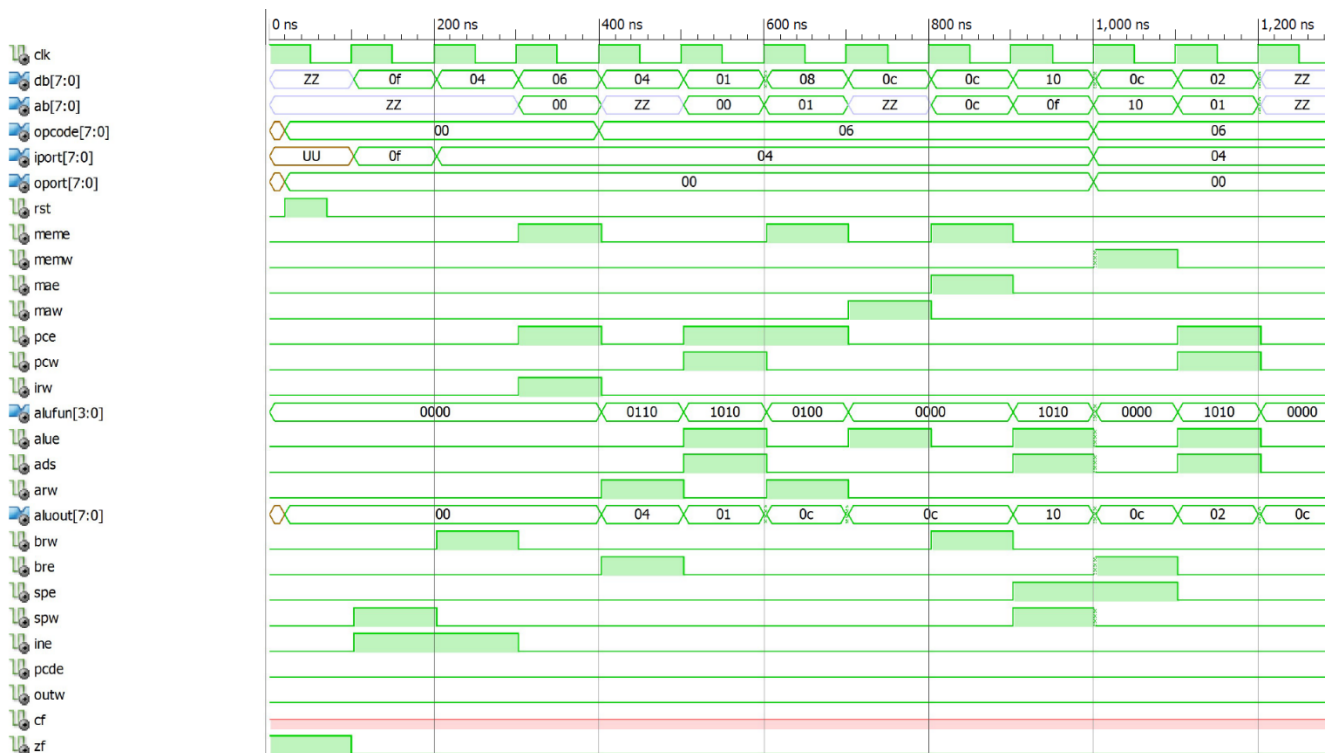
RTL Beschreibung: $\text{Stack} \leftarrow \text{MEM}[\text{BR} + x]$

3.2 Mikrooperationen

1: PCe, MEMe, IRw:	$IR \leftarrow MEM[PC]$
2: BRe, ADs, alufun, ARw:	$AKKU \leftarrow BR$
3: PCe, ADs, ALUe, alufun, PCw:	$PC \leftarrow PC + 1$
4: PCe, MEMe, ADs, alufun, ARw:	$AKKU \leftarrow AKKU + MEM[PC]$
5: ALUe, alufun, MAw:	$MAR \leftarrow AKKU$
6: MAe, MEMe, BRw:	$BR \leftarrow MEM[MAR]$
7: SPe, ADs, alufun, ALUe, SPw:	$SP \leftarrow SP + 1$
8: SPe, BRe, MEMw:	$MEM[SP] \leftarrow BR$
9: PCe, ADs, ALUe, alufun, PCw:	$PC \leftarrow PC + 1$

3.3 Simulation

Aus Platzgründen hier zuerst die Simulation, der Code der Test-Bench folgt auf der nächsten Seite.



In den ersten beiden Takten wurde der Stack-Pointer auf 0x0F gesetzt und das BR-Register auf 0x04. Im Memory befand sich nach dem OP-Code (Adresse 0x01) der Wert 0x08 (=Operand). Dies ergab die endgültige Adresse 0x0C. Zufällig war im Memory auf dieser Adresse ebenfalls der Wert 0x0C gespeichert. Der Transfer auf den Stack funktionierte problemlos.

3.4 VHDL Test-Bench Code

Hier wieder der relevante Code:

```
stim_p: process
    Begin

        wait for 103 ns; -- -1. Takt
MEMw<='0';MAw<='0';MAe<='0';MEMe<='0';ALUe<='0';ADs<='0';ARw<='0';alufun<="0000";PCw<='0';
PCe<='0';PCde<='0';SPe<='0';INe<='0';OUTw<='0';IRw<='0';SPw<='0';BRw<='0';BRe<='0'; -- Alle
auf Null setzen

        Iport <= x"0F";
        INe<='1'; SPw<='1'; -- Stackpointer auf 0xF setzen

        wait for 100 ns; -- 0. Takt
-- Alle auf Null setzen

        Iport <= x"04";
        INe<='1'; BRw<='1'; -- BR Register auf 0x4 setzen

        wait for 100 ns; -- 1. Takt
-- Alle auf Null setzen

        PCe<='1'; MEMe<='1'; IRw<='1'; --IR <- MEM[PC]

        wait for 100 ns; -- 2. Takt
-- Alle auf Null setzen

        BRe<='1'; ADs<='0'; alufun<="0110"; ARw<='1'; --AKKU <- BR

        wait for 100 ns; -- 3. Takt
-- Alle auf Null setzen

        PCe<='1'; ADs<='1'; ALUe<='1'; alufun<="1010"; PCw<='1'; --PC <- PC + 1

        wait for 100 ns; -- 4. Takt
-- Alle auf Null setzen

        PCe<='1'; MEMe<='1'; ADs<='0'; alufun<="0100"; ARw<='1'; --AKKU <- AKKU + MEM[PC]

        wait for 100 ns; -- 5. Takt
-- Alle auf Null setzen

        ALUe<='1'; alufun<="0000"; MAw<='1'; --MAR <- AKKU

        wait for 100 ns; -- 6. Takt
-- Alle auf Null setzen

        MAe<='1'; MEMe<='1'; BRw<='1'; --BR <- MEM[MAR]

        wait for 100 ns; -- 7. Takt
-- Alle auf Null setzen

        SPe<='1'; ADs<='1'; alufun<="1010"; ALUe<='1'; SPw<='1'; --SP <- SP + 1

        wait for 100 ns; -- 8. Takt
-- Alle auf Null setzen

        SPe<='1'; BRe<='1'; MEMw<='1'; --MEM[SP] <- BR

        wait for 100 ns; -- 9. Takt
-- Alle auf Null setzen

        PCe<='1'; ADs<='1'; ALUe<='1'; alufun<="1010"; PCw<='1'; --PC <- PC + 1

        wait for 100 ns; -- 10. Takt
-- Alle auf Null setzen

        wait for 100 ns; -- 11. Takt

    End process;
```

Wieder wurde das Nullsetzen der Signale nach jedem Takt aus Platzgründen durch Kommentare ersetzt.

4 Sprung-Befehl

4.1 Beschreibung

Gewählter Befehl: JPINDBR

Beispiel: JPINDBR

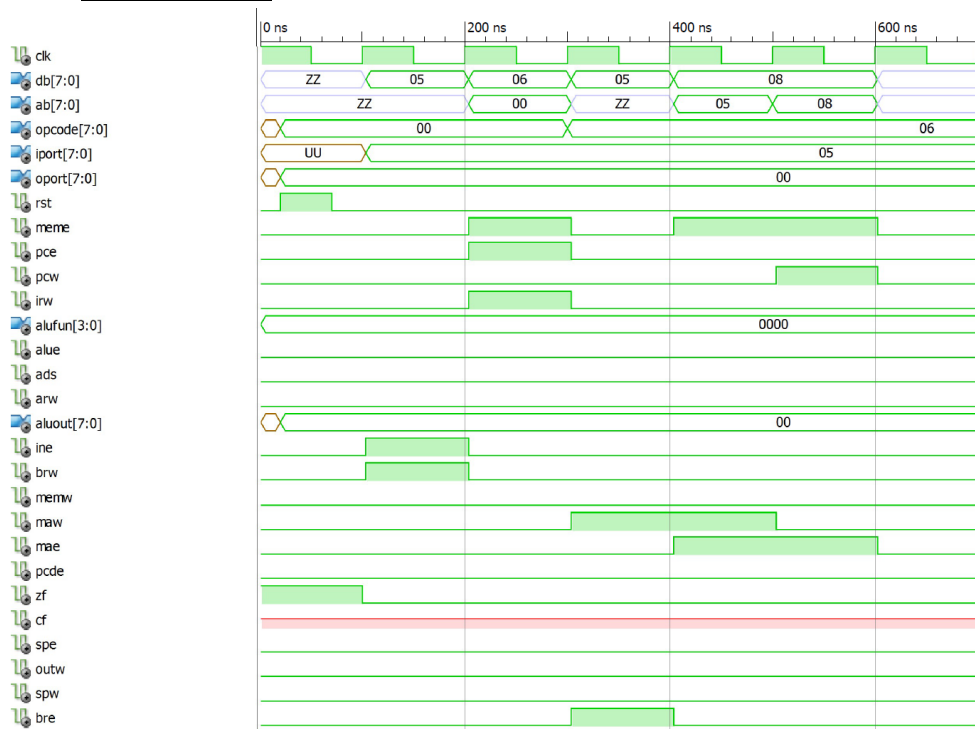
Dieser Befehl soll einen Sprung auf eine Adresse ausführen, welche sich im Memory auf einer Speicheradresse befindet. Diese Speicheradresse befindet sich wiederum im BR-Register (Indirekte Adressierung).

RTL Beschreibung: $PC \leftarrow MEM[BR]$

4.2 Mikrooperationen

- | | |
|--------------------|---------------------------|
| 1: PCe, MEMe, IRw: | $IR \leftarrow MEM[PC]$ |
| 2: BRe, MAw | $MAR \leftarrow BR$ |
| 3: MAe, MEMe, MAw: | $MAR \leftarrow MEM[MAR]$ |
| 4: MAe, MEMe, PCw: | $PC \leftarrow MEM[MAR]$ |

4.3 Simulation



Im ersten Takt wird 0x05 in das BR-Register geschrieben. In der Speicheradresse 0x05 befand sich in diesem Fall der Wert 0x08. Zufälligerweise war der Inhalt der Speicherzelle 0x08 ebenfalls 0x08 und somit auch die Jump-Adresse. Wie zu sehen ist funktionierte auch dieser Befehl problemlos.

4.4 VHDL Test-Bench Code

```
stim_p: process
  Begin

    wait for 103 ns; -- 0. Takt
    MEMw<='0';MAw<='0';MAe<='0';MEMe<='0';ALUe<='0';ADs<='0';ARw<='0';alufun<="0000";PC
    w<='0';PCe<='0';PCde<='0';SPe<='0';INe<='0';OUTw<='0';IRw<='0';SPw<='0';BRw<='0';BR
    e<='0'; -- Alle auf Null setzen

    Iport <= x"05";
    INe<='1'; BRw<='1'; -- BR Register auf 0x5 setzen

    wait for 100 ns; -- 1. Takt
    -- Alle auf Null setzen

    PCe<='1'; MEMe<='1'; IRw<='1'; --IR <- MEM[PC]

    wait for 100 ns; -- 2. Takt
    -- Alle auf Null setzen

    BRE<='1'; MAw<='1'; -- MAR <- BR

    wait for 100 ns; -- 3. Takt
    -- Alle auf Null setzen

    MAe<='1'; MEMe<='1'; MAw<='1'; -- MAR <- MEM[MAR]

    wait for 100 ns; -- 4. Takt
    -- Alle auf Null setzen

    MAe<='1'; MEMe<='1'; PCw<='1'; -- PC <- MEM [MAR]

    wait for 100 ns; -- 5. Takt

    MEMw<='0';MAw<='0';MAe<='0';MEMe<='0';ALUe<='0';ADs<='0';ARw<='0';alufun<="0000";PC
    w<='0';PCe<='0';PCde<='0';SPe<='0';INe<='0';OUTw<='0';IRw<='0';SPw<='0';BRw<='0';BR
    e<='0';

    wait for 400 ns; -- 6.-9. Takt

  End process;
```

Das Null setzen der Steuersignale nach jedem Takt wurde wieder durch Kommentare ersetzt. Bei Takt 0 ist ersichtlich, wie das BR-Register vor dem IF. gesetzt wird.