Befehlssatz

PROL16

Version 1/2003

Thomas Klaus / Markus Schutti / Markus Lindorfer

Johannes Kepler Universität Linz
Institut für Integrierte Schaltungen
O. Univ.-Prof. Dr. Richard Hagelauer
Altenbergerstr. 69, A-4040 Linz/Österreich

1. Befehlssatzübersicht

1.1. Der PROL 16 Befehlssatz

Der Befehlsvorrat des PROL 16 gliedert sich in vier funktionale Gruppen:

- Datentransport
- arithmetische Operationen
- logische Operationen
- Programmverzweigung

1.1.1. Datentransport

Diese Operationen lassen sich in zwei Typen einteilen:

- Datentransport für allgemeine Zwecke
- Datentransport zwischen CPU und externen Speicher

Keine dieser Operationen beeinflußt eines der Flags.

Datentransport für allgemeine Zwecke

– MOVE bewirkt den Transport eines Registerwertes vom Quellen- zum Zielregister.
- LOADI transportiert ein Datenwort unmittelbar in ein Register.

Datentransport zwischen CPU und externen Speicher

- LOAD transportiert ein Datenwort aus dem Datenspeicher in ein Register, wobei der zweite Registeroperand auf die Adresse im Speicher zeigt.
- STORE transportiert einen Registerwert in den Datenspeicher, wobei der zweite Registeroperand auf die Adresse im Speicher zeigt.

1.1.2. Arithmetische Operationen

Der PROL 16 kennt zwei grundlegende mathematische Operationen. Das Übertragsbit CY erlaubt die Ausführung von Additionen und Subtraktionen ganzzahliger Binärzahlen mit und ohne Vorzeichen.

Addition

- INC (Inkrementieren) addiert eine Eins zum Registeroperanden indem auch das Ergebnis abgelegt wird.
- ADD (Addieren) addiert den zweiten Registeroperanden zum ersten Registeroperanden indem auch das Ergebnis abgelegt wird.

– ADDC (Addieren mit Übertrag) addiert den zweiten Registeroperanden und das Übertragsbit CY zum ersten Registeroperanden. Das Ergebnis wird im ersten Operanden abgelegt.

Subtraktion

- DEC (Dekrementieren) subtrahiert eine Eins zum Registeroperanden indem auch das Ergebnis abgelegt wird.
- SUB (Subtrahieren) subtrahiert den zweiten Registeroperanden vom ersten Registeroperanden indem auch das Ergebnis abgelegt wird.
- SUBC (Subtrahieren mit Übertrag) subtrahiert den zweiten Registeroperanden und das Übertragsbit CY vom ersten Registeroperanden. Das Ergebnis wird im ersten Operanden abgelegt.

Vergleich

- COMP (Compare) subtrahiert den zweiten Registeroperanden vom ersten Registeroperanden ohne Änderung der beiden Operanden.

Die Flags werden wie folgt beeinflußt.

Carry CY wird gesetzt, wenn die Operation einen Übertrag vom höchstwertigen Bit bzw. einen Übertrag vom niedrigstwertigen Bit ergibt.

Zero ZO wird gesetzt, wenn das Ergebnis der Operation 0 wird.

1.1.3. Logische Operationen

Operationen mit einem Operanden

- NOT ermöglicht bitweise Invertierung eines Registeroperanden.
- SHL ermöglicht bitweise Verschiebung eines Registeroperanden um ein Bit nach links.
- SHR ermöglicht bitweise Verschiebung eines Registeroperanden um ein Bit nach rechts.
- SHLC ermöglicht bitweise Verschiebung eines Registeroperanden inklusive dem Übertragsbit um ein Bit nach links.
- SHRC ermöglicht bitweise Verschiebung eines Registeroperanden inklusive dem Übertragsbit um ein Bit nach rechts.

Operationen mit zwei Operanden

- AND ermöglicht eine bitweise UND-Verknüpfung von zwei Registeroperanden, wobei das Ergebnis im ersten Operanden abgelegt wird.
- OR ermöglicht eine bitweise ODER-Verknüpfung von zwei Registeroperanden, wobei das Ergebnis im ersten Operanden abgelegt wird.

– XOR ermöglicht eine bitweise Exklusiv-ODER-Verknüpfung von zwei Registeroperanden, wobei das Ergebnis im ersten Operanden abgelegt wird.

1.1.4. Programmverzweigung

Diese Operationen gestatten eine Programmverzweigung zu einem nicht unmittelbar folgenden Speicherplatz, einige davon bei Vorliegen einer bestimmten Bedingung.

Absolute Sprünge

Diese Operationen führen zu einem Programmsprung vom gegenwärtigen Wert des Programmzählers zur Zieladresse.

– JUMP ermöglicht einen Sprung auf jene Programmadresse auf die der Registeroperand zeigt.

Bedingte Sprünge

Diese Befehle gestatten Sprünge, die von bestimmten Bedingungen abhängig sind.

– JUMPC erzeugt einen Sprung, wenn das Übertragsbit CY gesetzt ist.
– JUMPZ erzeugt einen Sprung, wenn das Nullbit ZO gesetzt ist.

Sonstige

– NOP führt keine Operation aus
– SLEEP terminiert das Programm indem der Programmzähler nicht mehr inkrementiert wird.

Befehle, welche die Flags beeinflussen

Befehl	Übertrag Carry CY	Null Zero ZO
ADD	X	X
ADDC	X	X
SUB	X	X
SUBC	X	X
COMP	X	X
INC	X	X
DEC	X	X
AND	0	X
OR	0	X
XOR	0	X
NOT	0	X
SHL	X	X
SHR	X	X
SHLC	X	X
SHRC	X	X

1.2. Zusammenfassung des Befehlsvorrats des PROL16

Datentransport

Mnemonic	Beschreibung	Bytes	Zyklen	Opcode (bin)
LOADI Ra, word	Transportiere unmittelbares Datenwort word zu Register Ra	2	3	000010
LOAD Ra, Rb	Transportiere indirekt (Register Rb=Adresse) aus RAM zum Register Ra	1	3	000011
STORE Ra, Rb	Transportiere Register Ra indirekt (Register Rb=Adresse) zum RAM	1	3	000100
MOVE Ra, Rb	Transportiere Register Rb zum Register Ra	1	2	001100

Arithmetische Operationen

Mnemonic	Beschreibung	Bytes	Zyklen	Opcode (bin)
ADD Ra, Rb	Addiere Register Rb zum Register Ra	1	2	010100
ADDC Ra, Rb	Addiere Register Rb und Übertrag zum Register Ra	1	2	010101
SUB Ra, Rb	Subtrahiere Register Rb von Register Ra	1	2	010110
SUBC Ra, Rb	Subtrahiere Register Rb und Übertrag von RegisterRa	1	2	010111
COMP Ra, Rb	Vergleiche (subtrahiere) Register Rb und Register Ra	1	2	011000
INC Ra	Inkrementiere Register Ra	1	2	011010
DEC Ra	Dekrementiere Register Ra	1	2	011011

Logische Operationen

Mnemonic	Beschreibung	Bytes	Zyklen	Opcode (bin)
AND Ra, Rb	Verknüpfe Register Rb durch logisches UND zum Register Ra	1	2	010000
OR Ra, Rb	Verknüpfe Register Rb durch logisches ODER zum Register Ra	1	2	010001
XOR Ra, Rb	Verknüpfe Register Rb durch logisches Exklusiv-ODER zum Register Ra	1	2	010010
NOT Ra	Invertiere Register Ra	1	2	010011
SHL Ra	Schiebe Register Ra nach links	1	2	011100
SHR Ra	Schiebe Register Ra nach rechts	1	2	011101
SHLC Ra	Schiebe Register Ra und Übertragsbit nach links	1	2	011110
SHRC Ra	Schiebe Register Ra und Übertragsbit nach rechts	1	2	011111

Programmverzweigung

Mnemonic	Beschreibung	Bytes	Zyklen	Opcode (bin)

JUMP Ra	Springe indirekt (Register Ra=Adresse)	1	2	001000
JUMPC Ra	Springe indirekt (Register Ra=Adresse), wenn Übertragsbit = 1	1	2	001010
JUMPZ Ra	Springe indirekt (Register Ra=Adresse), wenn Nullbit = 1	1	2	001011
NOP	Keine Operation	1	2	000000
SLEEP	Programmende	1	2	000001

2. Definition und Beschreibung der Befehle

2.1. Mnemonic Befehle

NOP

Funktion: Keine Operation

Beschreibung:
Die Programmausführung wird mit dem folgenden Befehl fortgesetzt. Mit Ausnahme des Programmzählers werden keine Register oder Flags beeinflußt.

Beispiel: Verzögerung um 6 Taktzyklen

```
NOP
NOP
NOP
```

Bytes: 1
Zyklen: 2

Operationsablauf: (PC) ← (PC) + 1

Befehlscodierung: 000000

SLEEP

Funktion: Programmende

Beschreibung:
Durch SLEEP wird die Programmausführung beendet. Der Programmzähler wird nicht mehr inkrementiert, sodaß kein weiterer Befehl abgearbeitet werden kann.

Beispiel:

```
SLEEP
```

Bytes: 1
Zyklen: 2

Operationsablauf: (PC) ← (PC)

Befehlscodierung: 000001

LOADI <Register> , <Word>

Funktion: Laden eines Registers mit einer Wortvariable

Beschreibung:
Die im zweiten Operanden angegebene Wortvariable wird in den im ersten Operanden genannten Register geladen. Die Wortvariable und andere Flags werden dabei nicht beeinflußt.

Beispiel:

```
LOADI R0, 1000h      ; R0 unmittelbar mit 1000h laden
LOADI R1, Adresse1 ; Adresse1 ist eine numerische Konstante oder Label
```

Bytes: 2
Zyklen: 3

Operationsablauf: (Register) ← Word
(PC) ← (PC) + 2

Befehlscodierung: 000010

LOAD <RegisterA> , <RegisterB>

Funktion: Laden eines Registers indirekt aus dem RAM

Beschreibung:
Das Wort an der durch den zweiten Operanden angegebenen Speicheradresse wird in den im ersten Operanden genannten Register geladen. Die Speicheradresse und andere Flags werden dabei nicht beeinflußt.

Beispiel:

```
LOADI R0, 1000h
LOAD  R1, R0          ; R1 mit Inhalt der Speicherstelle 1000h laden
```

Bytes: 1
Zyklen: 3

Operationsablauf: (RegisterA) ← ((RegisterB))
(PC) ← (PC) + 1

Befehlscodierung: 000011

STORE <RegisterA> , <RegisterB>

Funktion: Speichern eines Registers indirekt in ein RAM

Beschreibung:
Das durch den ersten Operanden angegebene Register wird an die durch den zweiten Operanden genannte Speicheradresse gespeichert. Die Speicheradresse und andere Flags werden dabei nicht beeinflußt.

Beispiel:

```
LOADI R0, 1000h
STORE R1, R0        ; R1 an Speicherstelle 1000h speichern
```

Bytes: 1
Zyklen: 3

Operationsablauf: ((RegisterB)) ← (RegisterA)
(PC) ← (PC) + 1

Befehlscodierung: 000100

JUMP <Register>

Funktion: Springe auf Programmadresse

Beschreibung:
Der Programmzähler wird mit der durch das Register angegebenen Programmadresse geladen.

Beispiel:

```
LOADI R0, 1000h
JUMP R0             ; Sprung auf 1000h
```

Bytes: 1
Zyklen: 2

Operationsablauf: (PC) ← (Register)

Befehlscodierung: 001000

JUMPC <Register>

Funktion: Springe auf Programmadresse, wenn Übertragsbit gesetzt ist

Beschreibung:
Wenn das Übertragsbit gesetzt ist, wird der Programmzähler mit der durch das Register angegebenen Programmadresse geladen. Andernfalls fährt das Programm mit dem nächsten Befehl fort. Das Übertragsbit wird nicht geändert.

Beispiel:

```
LOADI R0, 1000h
LOADI R1, 0FFFFh
INC R1          ; Carry wird gesetzt
JUMPC R0        ; Sprung auf 1000h
```

Bytes: 1
Zyklen: 2

Operationsablauf: FALLS (Carry) = 1
DANN (PC) ← (Register)
SONST (PC) ← (PC) + 1

Befehlscodierung: 001010

JUMPZ <Register>

Funktion: Springe auf Programmadresse, wenn Nullbit gesetzt ist

Beschreibung:
Wenn das Nullbit gesetzt ist, wird der Programmzähler mit der durch das Register angegebenen Programmadresse geladen. Andernfalls fährt das Programm mit dem nächsten Befehl fort. Das Nullbit wird nicht geändert.

Beispiel:

```
LOADI R0, 1000h
LOADI R1, 1h
DEC R1          ; Zero wird gesetzt
JUMPZ R0        ; Sprung auf 1000h
```

Bytes: 1
Zyklen: 2

Operationsablauf: FALLS (Zero) = 1
DANN (PC) ← (Register)
SONST (PC) ← (PC) + 1

Befehlscodierung: 001011

MOVE <RegisterA> , <RegisterB>

Funktion: Kopiere ein Register

Beschreibung:
Das im zweiten Operanden angegebene Register wird in den im ersten Operanden genannten Register kopiert. Das zweite Register wird dabei nicht geändert.

Beispiel:

```
LOADI R0, 1000h
MOVE R1, R0; R1 mit 1000h laden
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterB)
(PC) ← (PC) + 1

Befehlscodierung: 011000

AND <RegisterA> , <RegisterB>

Funktion: Verknüpfe Register durch logisches UND

Beschreibung:
Durch AND erfolgt eine bitweise logische UND-Operation zwischen den angegebenen Registern und eine Speicherung des Ergebnisses im ersten Operanden genannte Register. Das Übertragsbit wird gelöscht bzw. das Nullbit entsprechend beeinflußt.

Beispiel:

```
AND R0, R1
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) and (RegisterB)
(PC) ← (PC) + 1

Befehlscodierung: 010000

OR <RegisterA> , <RegisterB>

Funktion: Verknüpfe Register durch logisches ODER

Beschreibung:
Durch OR erfolgt eine bitweise logische ODER-Operation zwischen den angegebenen Registern und eine Speicherung des Ergebnisses im ersten Operanden genannte Register. Das Übertragsbit wird gelöscht bzw. das Nullbit entsprechend beeinflußt.

Beispiel:

```
OR R0, R1
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) or (RegisterB)
(PC) ← (PC) + 1

Befehlscodierung: 010001

XOR <RegisterA> , <RegisterB>

Funktion: Verknüpfe Register durch logisches XOR

Beschreibung:
Durch XOR erfolgt eine bitweise logische XOR-Operation zwischen den angegebenen Registern und eine Speicherung des Ergebnisses im ersten Operanden genannte Register. Das Übertragsbit wird gelöscht bzw. das Nullbit entsprechend beeinflußt.

Beispiel:

```
XOR R0, R1
XOR R0, R0          ; R0 löschen
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) xor (RegisterB)
(PC) ← (PC) + 1

Befehlscodierung: 010010

NOT <Register>

Funktion: Verknüpfe Register durch logisches NOT

Beschreibung:
Durch NOT erfolgt eine bitweise logische NOT-Operation des angegebenen Registers und eine Speicherung des Ergebnisses im selbigen. Das Übertragsbit wird gelöscht bzw. das Nullbit entsprechend beeinflußt.

Beispiel:

```
NOT R0
```

Bytes: 1
Zyklen: 2

Operationsablauf: (Register) ← not (Register)
(PC) ← (PC) + 1

Befehlscodierung: 010011

ADD <RegisterA> , <RegisterB>

Funktion: Addieren

Beschreibung:
ADD addiert das durch den zweiten Operanden angegebene Register zum Inhalt des im ersten Operanden genannte Register, wo auch das Ergebnis abgelegt wird. Das Übertragsbit wird gesetzt wenn sich ein Überlauf bei der Addition ergibt, das Nullbit wird gesetzt wenn das Ergebnis Null ist.

Beispiel:

```
LOADI R0, 0FFFFh
LOADI R1, 1h
ADD R0, R1          ; Carry und Zero werden gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) + (RegisterB)
(PC) ← (PC) + 1

Befehlscodierung: 010100

ADDC <RegisterA> , <RegisterB>

Funktion: Addiere mit Übertrag

Beschreibung:
ADDC addiert gleichzeitig das Übertragsbit und das durch den zweiten Operanden angegebene Register zum Inhalt des im ersten Operanden genannte Register, wo auch das Ergebnis abgelegt wird. Das Übertragsbit wird gesetzt wenn sich ein Überlauf bei der Addition ergibt, das Nullbit wird gesetzt wenn das Ergebnis Null ist.

Beispiel:

```
LOADI R0, 0FFFFh
INC R0              ; Carry setzen
LOADI R0, 0FFFFh
LOADI R1, 0
ADDC R0, R1 ; Carry und Zero werden gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) + (RegisterB) + (Carry)
(PC) ← (PC) + 1

Befehlscodierung: 010101

SUB <RegisterA> , <RegisterB>

Funktion: Subtrahieren

Beschreibung:
SUB subtrahiert das durch den zweiten Operanden angegebene Register vom Inhalt des im ersten Operanden genannte Register, wo auch das Ergebnis abgelegt wird. Das Übertragsbit wird gesetzt wenn sich ein Überlauf bei der Subtraktion ergibt, das Nullbit wird gesetzt wenn das Ergebnis Null ist.

Beispiel:

```
LOADI R0, 0
LOADI R1, 1
SUB R0, R1          ; Carry wird gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) - (RegisterB)
(PC) ← (PC) + 1

Befehlscodierung: 010110

SUBC <RegisterA> , <RegisterB>

Funktion: Subtrahiere mit Übertrag

Beschreibung:
SUB subtrahiert gleichzeitig das Übertragsbit und das durch den zweiten Operanden angegebene Register vom Inhalt des im ersten Operanden genannte Register, wo auch das Ergebnis abgelegt wird. Das Übertragsbit wird gesetzt wenn sich ein Überlauf bei der Subtraktion ergibt, das Nullbit wird gesetzt wenn das Ergebnis Null ist.

Beispiel:

```
LOADI R0, 0FFFFh
INC R0          ; Carry und Zero werden gesetzt
LOADI R1, 0
SUBC R0, R1     ; Carry bleibt gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf: (RegisterA) ← (RegisterA) - (RegisterB) - (Carry)
(PC) ← (PC) + 1

Befehlscodierung: 010111

COMP <RegisterA> , <RegisterB>

Funktion: Vergleiche

Beschreibung:
COMP vergleicht die Werte der in den beiden Operanden angegebenen Register, die beiden Operanden werden dabei nicht geändert. Das Übertragsbit wird gelöscht wenn der erste Operand größer oder gleich, gesetzt wenn der erste Operand kleiner als der zweite Operand ist. Sind beide Operanden gleich so wird das Nullbit gesetzt.

Beispiel:

```
LOADI R0, 0FFFFh
LOADI R1, 0FFFEh
COMP R0, R1 ; Carry wird gelöscht
INC R1
COMP R0, R1 ; Zero wird gesetzt
DEC R0
COMP R0, R1 ; Carry wird gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf:
FALLS (RegisterA) - (RegisterB) >= 0
DANN (Carry) ← 0
FALLS (RegisterA) - (RegisterB) < 0
DANN (Carry) ← 1
FALLS (RegisterA) - (RegisterB) = 0
DANN (Zero) ← 1

(PC) ← (PC) + 1

Befehlscodierung: 011000

INC <Register>

Funktion: Inkrementieren

Beschreibung:
Der Inhalt des angegebenen Registers wird um 1 inkrementiert. Das Übertragsbit bzw. Nullbit werden entsprechend beeinflußt.

Beispiel:

```
LOADI R0, 0FFFFh
INC R0              ; Carry und Zero werden gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf: (Register) ← (Register) + 1
(PC) ← (PC) + 1

Befehlscodierung: 011010

DEC <Register>

Funktion: Dekrementieren

Beschreibung:
Der Inhalt des angegebenen Registers wird um 1 dekrementiert. Das Übertragsbit bzw. Nullbit werden entsprechend beeinflußt.

Beispiel:

```
LOADI R0, 0
DEC R0              ; Carry wird gesetzt
```

Bytes: 1
Zyklen: 2

Operationsablauf: (Register) ← (Register) - 1
(PC) ← (PC) + 1

Befehlscodierung: 011011

SHL <Register>

Funktion: Schiebe Register nach links

Beschreibung:
Die 16 Bits des Registers werden um ein Bit nach links geschoben wobei Bit0 gelöscht wird und Bit15 dem Übertragsbit zugewiesen wird.

Beispiel:

```
LOADI R0, 1111_0000_0000_1111b
SHL R0              ; Carry setzen
```

Bytes: 1
Zyklen: 2

Operationsablauf: $(Register_{n+1}) \leftarrow (Register_n)$ n = 1 ... 15
$(Register_0) \leftarrow 0$
$(PC) \leftarrow (PC) + 1$

Befehlscodierung: 011100

SHR <Register>

Funktion: Schiebe Register nach rechts

Beschreibung:
Die 16 Bits des Registers werden um ein Bit nach rechts geschoben wobei Bit15 gelöscht wird und Bit0 dem Übertragsbit zugewiesen wird.

Beispiel:

```
LOADI R0, 1111_0000_0000_1111b
SHR R0              ; Carry setzen
```

Bytes: 1
Zyklen: 2

Operationsablauf: $(Register_n) \leftarrow (Register_{n+1})$ n = 0 ...14
$(Register_{15}) \leftarrow 0$
$(PC) \leftarrow (PC) + 1$

Befehlscodierung: 011101

SHLC <Register>

Funktion: Schiebe Register nach links durch Übertragsbit

Beschreibung:
Die 16 Bits des Registers und das Übertragsbit werden um ein Bit nach links geschoben, wobei das alte Übertragsbit dem Bit0 zugewiesen wird und das neue Übertragsbit das Bit15 erhält.

Beispiel:

```
LOADI R0, 1111_0000_0000_1111b
SHLC R0          ; Carry setzen
SHLC R0          ; Carry auf Bit0
```

Bytes: 1
Zyklen: 2

Operationsablauf: $(Register_{n+1}) \leftarrow (Register_n)$ n = 1 ... 15
$(Register_0) \leftarrow (Carry)$
$(PC) \leftarrow (PC) + 1$

Befehlscodierung: 011110

SHRC <Register>

Funktion: Schiebe Register nach rechts durch Übertragsbit

Beschreibung:
Die 16 Bits des Registers und das Übertragsbit werden um ein Bit nach rechts geschoben, wobei das alte Übertragsbit dem Bit15 zugewiesen wird und das neue Übertragsbit das Bit0 erhält.

Beispiel:

```
LOADI R0, 1111_0000_0000_1111b
SHRLC R0         ; Carry setzen
SHRLC R0         ; Carry auf Bit15
```

Bytes: 1
Zyklen: 2

Operationsablauf: $(Register_n) \leftarrow (Register_{n+1})$ n = 0 ... 14
$(Register_{15}) \leftarrow (Carry)$
$(PC) \leftarrow (PC) + 1$

Befehlscodierung: 011111