# Detailbeschreibung der arithmetisch-logischen Instruktionen der ARMv4 ISA

## Handout zum 8. Aufgabenblatt

Hardwarepraktikum SoSe 20

Die folgenden Hinweise ergänzen das entsprechende Aufgabenblatt sowie die bereits zur Verfügung gestellte Dokumentation, insbesondere die Übersicht der ARM-Befehlssatzarchitektur, sodass Sie erfolgreich die ALU eines ARM-Prozessors in VHDL realisieren können.

Die ALU wird für zahlreiche Berechnungen in unterschiedlichen ARM-Instruktionen benötigt, jedoch bilden die arithmetisch-logischen Instruktionen allein bereits die Obermenge aller auftretenden Operandenverknüpfungen. Es genügt für die Realisierung der ALU daher, nur diese 16 Instruktionen zu betrachten, deren Befehlsformat der Abbildung 1 entnommen werden kann. Die Bedeutung

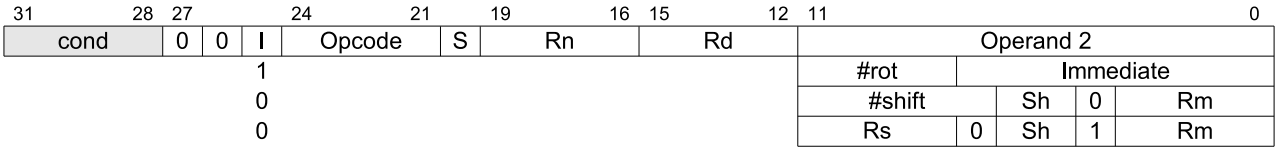


Abbildung 1: Befehlsformat der arithmetisch-logischen Instruktionen

der einzelnen Felder des Instruktionscodes wurde bereits in der Übersicht über die Befehlssatzarchitektur dargelegt. Für die Realisierung der ALU ist lediglich wichtig, dass Rn und Operand 2 Quelloperanden darstellen und Rd der Zieloperand ist. Woher Operand 2 genau stammt und dass er vor der ALU bereits einen Shifter durchlaufen haben kann, spielt für die ALU keine Rolle. Die 16 Arithmetisch-logischen Befehle unterscheiden sich lediglich durch den Wert des Opcode-Felds, welches in der Implementierung des HWPR-Prozessors direkt dem Steuereingang der ALU (`ALU_CTRL(3:0)`) entspricht.

Tabelle 1 gibt an, wie die Eingangsoperanden der ALU in Abhängigkeit vom Opcode-Feld verknüpft werden müssen. "`C`" entspricht dem Carry-Eingang der ALU (`ALU_CC_IN(1)`) und geht für einige Verknüpfungen in das Ergebnis mit ein. Beachten Sie, dass `C` für `SBC` und `RSC` *negiert* verwendet wird.

Die ALU erzeugt permanent einen vollständigen Ergebnisvektor (`RES`) und vollständige Statusbits (`CC`). Ob die Statusbits auch in das Statusregister übernommen werden und sich damit auf

| Opcode | Mnemonic | Bedeutung | Operation |
|---|---|---|---|
| 0000 | AND | AND | RES:= OP1 AND OP2 |
| 0001 | EOR | XOR | RES:= OP1 XOR OP2 |
| 0010 | SUB | Subtract | RES:= OP1 - OP2 |
| 0011 | RSB | Reverse Subtract | RES:= OP2 - OP1 |
| 0100 | ADD | Add | RES:= OP1 + OP2 |
| 0101 | ADC | Add with Carry | RES:= OP1 + OP2 + C |
| 0110 | SBC | Subtract with Carry | RES:= OP1 - OP2 - NOT C |
| 0111 | RSC | Reverse Subtract with Carry | RES:= OP2 - OP1 - NOT C |
| 1000 | TST | Test | CC:= OP1 AND OP2 |
| 1001 | TEQ | Test Equivalence | CC:= OP1 XOR OP2 |
| 1010 | CMP | Compare | CC:= OP1 - OP2 |
| 1011 | CMN | Compare Negated | CC:= OP1 + OP2 |
| 1100 | ORR | OR | RES:= OP1 OR OP2 |
| 1101 | MOV | Move | RES:= OP2 |
| 1110 | BIC | Bit Clear | RES:= OP1 AND NOT OP2 |
| 1111 | MVN | Move Not | RES:= NOT OP2 |

Tabelle 1: In der ALU realisierte Verknüpfungen und zugehörige Opcodes. Paarweise identische Verknüpfungen sind farblich hervorgehoben.

folgende Instruktionen auswirken, hängt von weiteren Randbedingungen ab. Das ALU-Ergebnis von Arithmetisch-logischen Instruktionen wird gewöhnlich in Register Rd zurückgeschrieben. Eine Ausnahme bilden die Instruktionen `TST`, `TEQ`, `CMP` und `CMN`. Sie sind paarweise identisch mit den Instruktionen `AND`, `EOR`, `SUB` und `ADD`, ihr Ergebnisvektor wird aber nicht in den Registerspeicher zurückgeschrieben. Stattdessen werden die erzeugten Statusbits unbedingt in das Statusregister übernommen (in Tabelle 1 angedeutet durch "`CC:=`").

Für die Erzeugung der Statusbits in der ALU gilt:

- N: *Negative*, Bit 31 des in der ALU ermittelten Ergebnisses.
- Z: *Zero,* wird gesetzt, wenn das ermittelte Ergebnis `RES = 0` ist.
- V: *Overflow*, Überlauf bei arithmetischen Operationen. Bei logischen Operationen verändert die ALU das V-Bit des Prozessors nicht, sondern reicht den Wert vom V-Eingang (`ALU_CC_IN(0)`) unverändert durch.
- C: *Carry*, bei allen arithmetischen Operationen wird das C-Bit durch die ALU gesetzt. Bei allen logischen Operationen wird das C-Bit unverändert durch die ALU geleitet. Bei allen arithmetischen Operationen, die eine Subtraktion beinhalten, wird das C-Bit gerade dann gesetzt, wenn während der Subtraktion kein Übertrag auftritt. Die betroffenen Operationen sind: `SUB`, `SBC`, `RSB`, `RSC`, `CMP`.

Für arithmetische Operationen gilt allgemein: Das Carry-Bit entspricht einem Übertrag bei der Addition oder Subtraktion über das MSB (das am höchsten wertige Bit) des Ergebnisses hinaus[1]. Seien $a_k$ und $b_k$ die k-ten Bits der Operanden A und B. Die bekannten Übertragsgleichungen für Stelle $k$ lauten dann:

[1] Subtraktionsoperationen im ARM-Prozessor setzen das Carry-Flag gerade dann, wenn kein Übertrag auftritt!

$$c_k = a_k \cdot b_k + c_{k-1} \cdot (a_k \oplus b_k) \qquad \text{(Addition)}$$
$$c_k = \overline{a_k} \cdot b_k + c_{k-1} \cdot (\overline{a_k} + b_k) \qquad \text{(Subtraktion)}$$

Für Addition und Subtraktion zweier $n$ Bit breiter Operanden lässt sich der Overflow bestimmen als:

$$v = c_n \oplus c_{n-1} \qquad \text{(Überlauf)}$$

Der Überlauf entspricht also der Exklusiv-Oder-Verknüpfung des Übertrags in die höchste Ergebnisstelle mit dem Übertrag über die höchste Ergebnisstelle hinaus (der gleichzeitig dem Carry-Bit der Addition entspricht und für ARM dem negierten Carry-Bit der Subtraktion). Bei der Realisierung von Addition oder Subtraktion in VHDL stehen die Überträge der einzelnen Ergebnisstellen nicht zur Verfügung, wenn die Arithmetikoperatoren "+" und "-" verwendet werden. Zur Ermittlung des Carry-Bits kann ggf. folgender Trick verwendet werden: Beide Operanden werden mit einer führenden Null auf 33 Bit erweitert, addiert/subtrahiert und das Ergebnis einem temporären Ergebnisvektor zugewiesen. Dessen Bits 31:0 entsprechen dem gesuchten Ergebnis der Addition/-Subtraktion, Bit 32 dem Übertrag.

C- und V-Bit können jedoch auch aus einer Betrachtung der Operanden und des 32-Bit-Ergebnisses abgeleitet werden, wenn die Bedeutung der Statusbits verstanden worden ist: Das Auftreten eines Übertrags (Carry) bedeutet, dass die Addition zweier vorzeichenlos interpretierter Zahlen der Breite $n$ zu einem Ergebnis außerhalb des darstellbaren Bereichs geführt hat. Ob das geschehen ist, lässt sich an den MSBs beider Summanden und des Ergebnisses ablesen:

- Sind beide Summanden-MSBs = 0, so kann das Ergebnis nicht außerhalb des darstellbaren Bereichs liegen. Für n = 4 und die größtmöglichen Summanden mit führender Null: 0111 + 0111 =1110
- Sind beide Summanden-MSBs = 1, so wird das Ergebnis immer außerhalb des darstellbaren Bereichs liegen, C muss hier also gesetzt werden: 1000 + 1000 != 0000
- Ist nur eines der Summanden-MSBs = 1, ist zusätzlich das Ergebnis heranzuziehen. Ist das MSB der Summe = 0, so muss ein Übertrag aufgetreten sein, denn die Summe kann nicht kleiner sein als einer der (positiven) Summanden.
- Die bei ARM auftretende Addition einer weiteren Eins (Add with Carry) ändert an diesen Überlegungen nichts.

Zusammengefasst gilt also:

$$c = a_n \cdot b_n + \overline{a_n} \cdot b_n \cdot \overline{z}_n + a_n \cdot \overline{b_n} \cdot \overline{z_n} \qquad \text{(Übertrag)}$$

wobei $z_n$ das MSB der berechneten Summe darstellt.

Der Überlauf (Overflow) zeigt an, dass bei der Addition zweier als vorzeichenbehaftet interpretierter Summanden das Ergebnis außerhalb des darstellbaren Bereichs liegt. Hier gilt: Sind entweder Augend oder Addend negativ, liegt das Ergebnis immer im darstellbaren Bereich, Sie können sich dies z.B. an einem Zahlenring für n = 4 verdeutlichen. Sind beide Summanden negativ, das Ergebnis aber

positiv, so ist dies offensichtlich falsch. Das gilt auch, wenn beide Summanden positiv sind und das Ergebnis negativ ist. In diesen Fällen muss das V-Bit gesetzt werden. Daraus folgt:

$$v = \overline{a_n} \cdot \overline{b_n} \cdot z_n + a_n \cdot b_n \cdot \overline{z_n} \qquad \text{(Überlauf)}$$

Auch hier beeinflusst die mögliche Addition einer weiteren Eins die Betrachtung nicht wesentlich. Ähnliche Herleitungen sind für die Subtraktion möglich.