Praktikum 06 - Assembler

-André Kirsch, Malte Riechmann

Aufgabe 1

Zu Beginn haben wir über `movs r4, %[num]` denn Wert von *number* in das Register r1 geladen. `[num]` ist ein Symbolischer Name für *number* und wurde hierfür in der Zeile `[num] "+r" (number)` als Output-Variable mit Lese- und Schreibberechtigung gekennzeichnet.

Um *number* zu verdoppeln führen wir auf den in das Register r1 geladenen Wert einen Bit shift nach links um ein Bit über den Befehl `lsl r4, r4, #1` aus. Um *number* wieder auf 1 zu setzen, vergleichen wir es mit 256. Da cmp quasi nur der subs Befehl ist, ohne das Ergebnis in ein Register zu schreiben, wird, wenn der Wert von r1 und 256 gleich sind, das Z Flag auf 1 gesetzt. Ob das Z Flag = 1 ist wird über das Suffix eq abgefragt. Durch die Zeile `it eq` wird die darauffolgende Zeile `moveq r4, #1`, um den Wert 1 in das Register r4 zu schreiben, nur Ausgeführt, wenn Z = 0 ist.

Am Ende des Assembler Block steht zuerst die Liste der Output Operanden, gefolgt von den Input Oberanden. Am Ende steht die *clobber list*:

r4 - Kennzeichnung für den Compiler, dass Register r4 modifiziert wird. Dieser speichert den darinstehenden Wert und lädt die gespeicherten Wert am Ende des Blocks wieder in das Register.

cc - Kennzeichnung für den Compiler, dass die condition flags modifiziert werden.

memory - Kennzeichnung für den Compiler, dass der Assembler-Block die Speicherorte ändern kann und zwingt ihn dazu alle gecacheten Wert zu speichern und anschließend wieder zu laden.

Taktzyklen:

Anweisung	Zyklen
mov (3x)	1
lsl	1
cmp	1
it	1
Zyklen maximal	6
Zyklen minimal	5

Bei den Taktzyklen werden nicht durch den Compiler eventuell hinzugefügt load oder store Befehle zur Verarbeitung der In- und Output Variablen berücksichtigt

```
asm volatile(
  "movs r4, %[num]\n\t"
  "lsl r4, r4, #1\n\t"
  "cmp r4, #256\n\t"
  "it eq\n\t"
  "moveq r4, #1\n\t"
  "movs %[num], r4 \n\t"
: [num] "+r" (number)
:
: "r4", "cc", "memory"
);
```

Aufgabe 2

Zuerst laden wir die Speicherstelle des *fiboData* Arrays in das Register r4. Dazu verwenden wir den Befehl `movs r4, %[fibo]`. *Fibo* haben wir als Input Variable am Ende des Assembler Codes angegeben: `[fibo] "r" (fiboData)`. Des Weiteren laden wir auch den *lastFiboIndex* in ein Register: `movs r7, %(index)`. Auch diesen haben wir analog zur *fibo* Variable angegeben. Da schon die ersten zwei Werte in das Array eingetragen sind, können wir die ersten zwei Durchläufe überspringen. Dies tun wir mit dem Befehl `subs r7, #2`. Den restlichen Verlauf des Assembler Abschnittes haben wir in die Funktion `fibonacci:` zusammengefasst. Darin laden wir zuerst die Werte an der ersten und zweiten des Arrays abhängig von der aktuellen Position des Zeigers in die Register r5 und r6: `ldr r5, [r4, #0]` `ldr r6, [r4, #1]`. Danach addieren wir die Werte aus r5 und r6: `add r5, r6` und schreiben das Ergebnis wieder zurück nach r5. Danach schreiben wir r5 zurück in den Array in die nächste freie Stelle `str r5, [r4, #2]`. Anschließend schieben wir den Array um eine Stelle weiter: `add r4, #1` und verringern die *index*-Variable um eins: `subs r7, #1`. Dabei werden gleichzeitig die Flags gesetzt. Der letzte Befehl `bne fibonacci` ist ein bedingter branch Befehl. Dabei springt er jedes Mal zum Beginn der fibonacci Funktion, solange das zero-Flag nicht gesetzt wurde bzw. in Kombination mit dem vorherigen Befehl der Index noch nicht null erreicht hat. Die Input und Output Variablen und die Clobber List sind Analog zu Aufgabe 1 mit mehr Registern. Die Aufzählung der genutzten Registern in der Clobber List sorgt dafür, dass die Register nach der Durchführung ihre ursprünglichen Werte bekommen.

Taktzyklen:

Anweisung	Zyklen
mov (2x)	1
sub (2x)	1
ldr (2x)	2
str (1x)	2
b (1x)	1 (+ P) -> +P nur wenn der Sprung durchgeführt wird, wobei P 1-3 groß sein kann.
add (2x)	1
Gesamt	3 + 10 * (lastFiboIndex - 2) + P * (lastFiboIndex - 3) -> in diesem Fall: 3 + 10 * 11 + P * 10 = 113 + 10P

```
asm volatile(
  "movs r4, %[fibo]\n\t"
  "movs r7, %[index]\n"
  "subs r7, #2\n\t"
  "fibonacci:\n\t"
  "ldr r5, [r4, #0]\n\t"
  "ldr r6, [r4, #1]\n\t"
  "add r5, r6\n\t"
  "str r5, [r4, #2]\n\t"
  "add r4, #1\n\t"
  "subs r7, #1\n\t"
  "bne fibonacci"
  :
  : [index] "r" (lastFiboIndex), [fibo] "r" (fibData)
  : "r4", "r5", "r6", "r7", "cc", "memory"
);
```