

Prof. L. Thiele

## Technische Informatik - FS 2021

**Musterlösung zu Übung 1**

Datum : 04.-05. März 2021

**Aufgabe 1: Instruktionssatz****Aufgabe 1.1: Speicheradressierung**

Nehmen Sie an, das Register \$gp enthält den Wert  $10008000_{hex}$ . Zum Lesen eines Wertes aus dem Hauptspeicher wird der Befehl

$$lw \quad \$t0, x(\$gp)$$

verwendet, wobei x eine Zahl bedeutet.

- (a) Welche kleinste Hauptspeicheradresse kann gelesen werden? Wie lautet die Instruktion?

**Lösung:**

Die kleinste Hauptspeicheradresse, die gelesen werden kann ist  $10000000_{hex}$ .

$$lw \quad \$t0, -32768(\$gp)$$

- (b) Welche grösste Hauptspeicheradresse kann gelesen werden? Wie lautet die Instruktion?

**Lösung:**

Die grösste Hauptspeicheradresse, die codiert werden könnte, wäre  $1000FFFF_{hex}$ . Die resultierende Adresse der  $lw$  Instruktion muss allerdings an Wortgrenzen ausgerichtet sein. Deshalb ist das höchste Wort, welches mit einer gültigen Instruktion adressiert werden kann jenes an Adresse  $1000FFFC_{hex}$ . Dabei wird unter anderem auch das Byte an Adresse  $1000FFFF_{hex}$  mitgelesen.

$$lw \quad \$t0, 32764(\$gp)$$

- (c) Wie lautet die Instruktion falls die Adresse  $10006EDC_{hex}$  gelesen werden soll?

**Lösung:**

Der Offset zum Zielregister ist:  $6EDC_{hex} - 8000_{hex} = (-1) \cdot 1124_{hex} = -4388_{dec}$

$$lw \quad \$t0, -4388(\$gp)$$

- (d) Wie lauten die Instruktionen in (a)-(c) in binärer Maschinensprache?

**Lösung:**

Der Befehl lw hat die Operationskodierung (OP) 35.

Das Quellenregister (RS) ist \$gp mit Registernummer 28.

Das Zielregister (RT) ist \$t0 mit Registernummer 8

Aufgabe	OP	RS	RT	immediate
(a)	100011	11100	01000	1000 0000 0000 0000
(b)	100011	11100	01000	0111 1111 1111 1100
(c)	100011	11100	01000	1110 1110 1101 1100

Für c. ergibt sich der Wert durch die Umwandlung von  $1124_{hex}$  in Binärform und anschliessender Negation durch das Zweierkomplement.

$$1124_{hex} = 0001000100100100_{bin} \xrightarrow{\text{Zweierkomplement}} 1110111011011100_{bin}$$

- (e) Nehmen Sie an, das Register \$gp wurde noch nicht initialisiert. Schreiben Sie eine Sequenz von Instruktionen, an deren Ende das Register \$gp den Wert  $10008000_{hex}$  hat.

**Lösung:**

```
lui    $gp, 0x1000
ori    $gp, $gp, 0x8000
```

**Aufgabe 1.2: Instruktionskodierung**

Nehmen Sie an, die erste Instruktion eines Programms steht an der Hauptspeicheradresse  $00400000_{hex}$ . Der Hauptspeicher enthält die folgenden Werte:

Adresse (hex)	Wert (bin)	Adresse (hex)	Wert (bin)
0040 0000	0011 0100	0040 0006	1001 0011
0040 0001	0001 0001	0040 0007	1100 0000
0040 0002	0100 0000	0040 0008	1010 1110
0040 0003	0000 0001	0040 0009	0101 0010
0040 0004	0000 0000	0040 000a	1111 0000
0040 0005	0001 0001	0040 000b	0000 0000

An welchen Adressen im Hauptspeicher werden Werte verändert und welche Werte stehen dort nach Ausführung des Programms?

**Lösung:**

Die Werte des Hauptspeicher können zu Wörtern aus je 32 Bit zusammengefasst und dann mit Hilfe der Folien 2-18, 2-28, 2-29 und 2-32 zu Instruktionen umgeschrieben werden.

OP	RS	RT	immediate
001101	00000	10001	0100 0000 0000 0001

  

OP	RS	RT	RD	shamt	funct
000000	00000	10001	10010	01111	000000

OP	RS	RT	immediate
101011	10010	10010	1111 0000 0000 0000

Daraus folgen die Assemblerbefehle:

```
ori    $s1, $zero, 0x4001
sll    $s2, $s1, 15
sw     $s2, -4096($s2)
```

Die sw Instruktion schreibt den Wert von \$s2 an die Adresse in \$s2 mit Offset -4096. Der Wert in \$s2 ist  $20008000_{hex}$ . Mit Offset ergibt sich die Adresse  $20007000_{hex}$  und die folgende Speicherbelegung:

Adresse (hex)	Wert (bin)
2000 7000	0010 0000
2000 7001	0000 0000
2000 7002	1000 0000
2000 7003	0000 0000

### Aufgabe 1.3: Synchronisation

Zwei Prozessoren greifen auf einen einzigen Hauptspeicher zu. Falls sie gleichzeitig schreiben (sw) oder lesen (lw), so ist die Reihenfolge der Ausführung zufällig.

Es soll der Term  $n(n+1)/2$  ausgerechnet werden und zwar durch aufaddieren von  $1 + 2 + 3 \dots + n$ . Die beiden Prozessoren benötigen sich verändernde, sehr unterschiedliche und unbekannte Ausführungszeiten pro Instruktion. Um eine schnelle Ausführung zu erreichen, wird die gleiche Aufgabe beiden Prozessoren gestellt. Derjenige Prozessor, der gerade schneller ist, 'hilft' dem Anderen über die Kommunikation von Zwischenergebnissen. Am Ende sollen beide Prozessoren das Ergebnis in einem Register speichern.  $n$  ist am Anfang in beiden Prozessoren jeweils im Register \$s0. Das Ergebnis soll am Ende in beiden Prozessoren jeweils im Register \$t1 stehen. Die Prozessoren sollen zur Kommunikation einen Speicherbereich nutzen, dessen Startadresse bei Programmbeginn in dem Register \$s3 steht.

- (a) Am Anfang sind alle Werte im Hauptspeicher gleich 0. Schreiben Sie für die beiden Prozessoren jeweils eine Sequenz von Instruktionen, so dass der Term  $n(n+1)/2$  wie oben beschrieben berechnet wird.

#### Lösung:

Zum besseren Verständnis wird das Programm zuerst in Pseudo-Code geschrieben:

```
t1 = 0           // Summe
s1 = 0           // Zähler
while s1 < s0 do
    s1 = s1 + 1
    t1 = t1 + s1
    wait(lock)
    s2 = memory1
    if s1 < s2 then
        s1 = s2
        t1 = memory2
    else if s2 < s1 then
        memory1 = s1
```

```

        memory2 = t1
    fi
    signal(lock)
od

```

Im zweiten Schritt kann der Pseudo-Code in ein Assembler Programm umgeschrieben werden

```

        add    $t1, $zero, $zero    # Initialisiere t1
        add    $s1, $zero, $zero    # Initialisiere s1
loop:
        beq    $s1, $s0, loop_end   # while s1 != s0
        addi   $s1, $s1, 1          # Inkrementiere Zähler
        add    $t1, $t1, $s1        # Berechne die Summe
try_lock:
        ll     $t3, 0($s3)           # ll auf die Variable lock
        bne    $t3, $zero, try_lock  # Wiederhole bis lock == 0
        addi   $t2, $zero, 1         # $t2 = 1
        sc     $t2, 0($s3)           # sc auf Variable lock
        beq    $t2, $zero, try_lock  # Falls sc nicht erfolgreich -> try_lock

        # Jetzt kann der Speicherbereich sicher genutzt werden
        lw     $s2, 4($s3)           # Lade den Zähler aus dem Speicher

        # if s1 < s2
        slt    $t2, $s1, $s2        # Überprüfe welcher Prozess weiter ...
        beq    $t2, $zero, elseif    # ... fortgeschritten ist
        # s1 ist kleiner als s2
        lw     $t1, 8($s3)           # Aktualisiere interne Register ...
        addi   $s1, $zero, $s2       # ... mit den Werten aus dem Speicher
        j      unlock

elseif: # if s2 > s1
        slt    $t2, $s2, $s1
        beq    $t2, $zero, unlock
        # s2 ist kleiner als s1
        sw     $s1, 4($s3)           # Überschreibe die Werte im Speicher
        sw     $t1, 8($s3)

        # Im Falle von s2 == s1 ist der Speicherzugriff nicht nötig

unlock:
        sw     $zero, 0($s3)         # Setze lock = 0
        j      loop                  # Springe zum Beginn der Schleife
loop_end:

```

- (b) Geben Sie, falls möglich, eine Lösung an, die auch korrekt ist, falls der Datenspeicher nicht mit dem Wert 0 vorbesetzt ist.

### Lösung:

Wenn der Datenspeicher nicht mit 0 initialisiert ist kann eine Speicherzugriffkollision nicht ausgeschlossen werden. Es kann nicht eindeutig bestimmt werden, ob die lock Variable gesetzt ist oder nicht.