Datenpfaderweiterung

Der Single Cycle Datenpfad des MIPS Prozessors soll um die Instruktion

```
min $t0, $t1, $t2
```

erweitert werden, welche den kleineren der beiden Registerwerte `$t1` und `$t2` in einem Zielregister `$t0` speichert.

1. Wie sieht eine MIPS-Assembler Implementierung dieser Instruktion ohne Pseudo-Befehle aus?
2. Welches Befehlsformat würden Sie für die neue Instruktion wählen? Geben Sie die Belegung der einzelnen Befehlsfelder an.
3. Erweitern Sie die den Datenpfad um die benötigte Hardware, damit die `min(a,b)` Instruktion unterstützt wird. Sie dürfen dafür alle in der Vorlesung vorgestellten und im Datenpfad enthaltenen Gatter und Komponenten verwenden.
4. Geben Sie die Werte der Steuersignale an, damit der `min(a,b)` Befehl ausgeführt wird. Verwenden Sie *don't care* wenn möglich.

Datenpfaderweiterung

1. Wie sieht eine MIPS-Assembler Implementierung dieser Instruktion ohne Pseudo-Befehle aus?

```
min:    slt $t0, $t1, $t2          \# $t1 < $t2?
        beq $0, $t0, else
        add $t0, $0, $t1           \# $t0 = $t1
        j end
else:   add $t0, $0, $t2           \# $t0 = $t2
end:    ...
```

Datenpfaderweiterung

2. Wie sieht eine MIPS-Assembler Implementierung dieser Instruktion ohne Pseudo-Befehle aus?

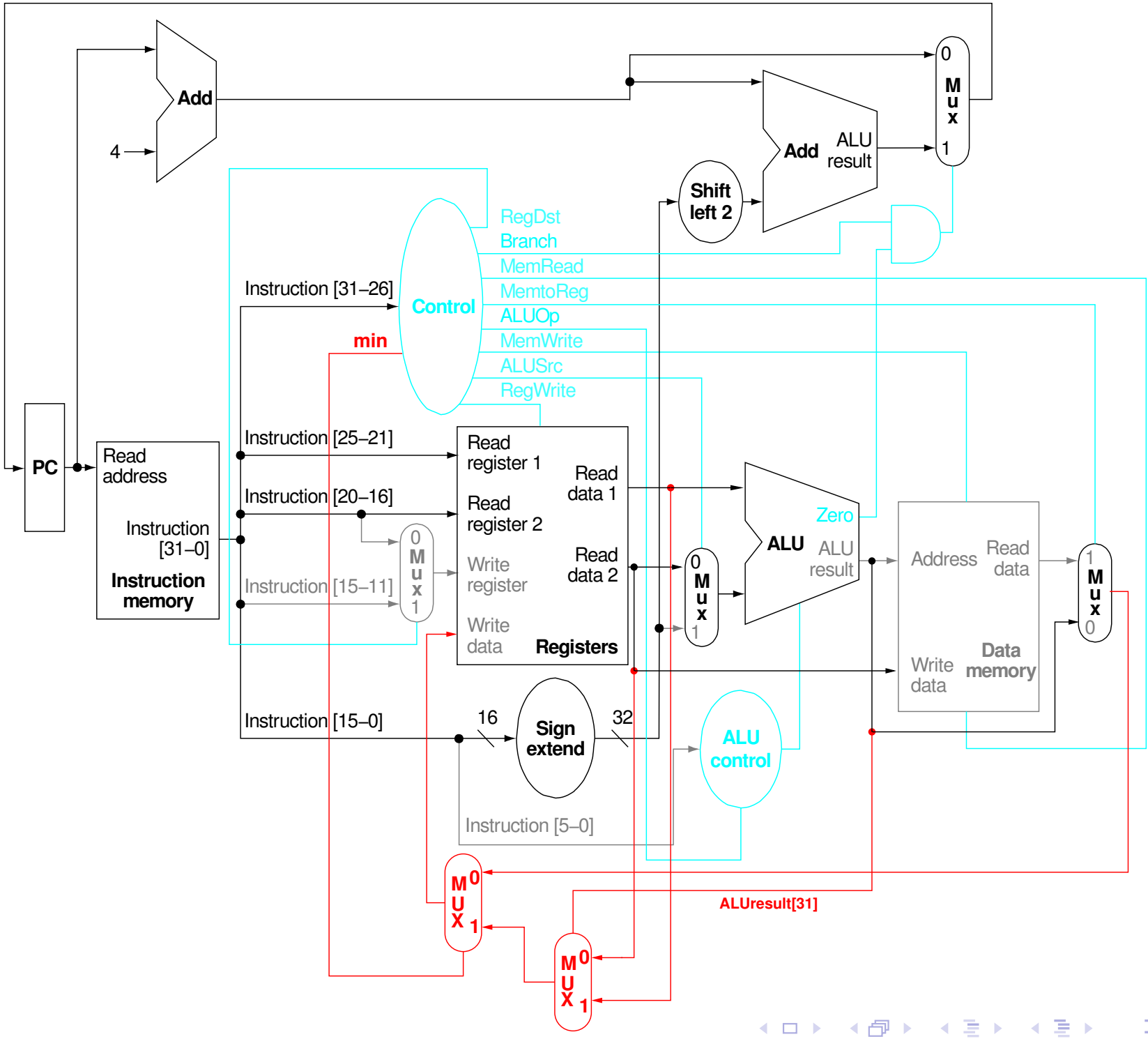
R-Typ, da der Befehl drei Operanden hat:

0	$t1	$t2	$t0	0	0x03*

*Der funct-Code muss so gewählt werden, dass er noch nicht für andere Operationen belegt ist, hier ist also eine Vielzahl an Lösungen möglich.

Datenpfaderweiterung

3. Erweitern Sie die den Datenpfad um die benötigte Hardware, damit die `min(a,b)` Instruktion unterstützt wird. Sie dürfen dafür alle in der Vorlesung vorgestellten und im Datenpfad enthaltenen Gatter und Komponenten verwenden.


Add
4
0
M
u
x
1
Add
ALU
result
Shift
left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
min
PC
Read
address
Instruction
[31–0]
Instruction
memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
16
Sign
extend
32
ALU
Zero
ALU
result
ALU
control
Address
Read
data
Write
data
Data
memory
MUX
ALUresult[31]

Datenpfaderweiterung

4. Geben Sie die Werte der Steuersignale an, damit der `min(a,b)` Befehl ausgeführt wird. Verwenden Sie *don't care* wenn möglich.

Steuersignale müssen wie bei einem standard R-Typ Befehl gesetzt sein + neuem Steuersignal; genau genommen müsste MemToReg auf x gesetzt werden, da der Befehl aber in Ctrl nur als R-Typ gesehen wird, kann das auch nicht umgesetzt werden. Lösung siehe Datenpfad.

Pipelining

Gegeben ist folgender MIPS-Assembler Code:

```
1  add   $t1, $t2, $t3
2  sub   $t4, $t2, $t5
3  or    $t6, $t1, $t4
4  and   $t7, $t8, $t9
5  lw    $t9, 4($t7)
6  sub   $t2, $t1, $t3
7  lw    $t1, 16($t9)
```

Angenommen, der Code soll auf dem MIPS Pipelined Prozessor **ohne** Forwarding Unit und Hazard Detection ausgeführt werden.

1. Identifizieren und benennen Sie alle möglich auftretenden Hazards.
2. Fügen Sie `NOP` Operationen in den Code ein, um Hazards zu verhindern.
3. Minimieren Sie die Anzahl der notwendigen `NOP`s durch Veränderung der Codereihenfolge, ohne die Funktionalität des Codes zu beeinflussen.
4. Wie groß ist der CPI des Code-Abschnitts vor und nach Ihrer Optimierung? Gehen Sie davon aus, dass die Pipeline zu Beginn der Code-Ausführung leer ist. Welchen Speed-Up konnten Sie erzielen?

Pipelining

```
1   add             $t1, $t2, $t3
2   sub             $t4, $t2, $t5
3   or              $t6, $t1, $t4
4   and             $t7, $t8, $t9
5   lw              $t9, 4($t7)
6   sub             $t2, $t1, $t3
7   lw              $t1, 16($t9)
```

1. Identifizieren und benennen Sie alle möglich auftretenden Hazards.

- 1. -> 3. $t1: data hazard
- 2. -> 3. $t4: data hazard
- 4. -> 5. $7: data hazard
- 5. -> 7. $t9: load-use hazard (data hazard)

Pipelining

```
1   add             $t1, $t2, $t3
2   sub             $t4, $t2, $t5
3   or              $t6, $t1, $t4
4   and             $t7, $t8, $t9
5   lw              $t9, 4($t7)
6   sub             $t2, $t1, $t3
7   lw              $t1, 16($t9)
```

2. Fügen Sie `NOP` Operationen in den Code ein, um Hazards zu verhindern.

```
1   add
2   sub
3   NOP
4   NOP
5   or
6   and
7   NOP
8   NOP
9   lw
10  sub
11  NOP
12  lw
```

Letztes NOP kann vor oder nach dem `sub` platziert werden.

Pipelining

3. Minimieren Sie die Anzahl der notwendigen `NOP`s durch Veränderung der Codereihenfolge, ohne die Funktionalität des Codes zu beeinflussen.

```
1  add   $t1, $t2, $t3
2  sub   $t4, $t2, $t5
3  or    $t6, $t1, $t4
4  and   $t7, $t8, $t9
5  lw    $t9, 4($t7)
6  sub   $t2, $t1, $t3
7  lw    $t1, 16($t9)
```

```
and   $t7, $t8, $t9
add   $t1, $t2, $t3
sub   $t4, $t2, $t5
lw    $t9, 4($t7)
sub   $t2, $t1, $t3
or    $t6, $t1, $t4
lw    $t1, 16($t9)
```

- Zeile 4 und 5 sind von den ersten drei Zeilen unabhängig und können benutzt werden, um NOPs zu entfernen. Da zwischen 4 und 5 zwei Befehle liegen müssern, “umklammern” Zeile 4 und 5 nun die Zeilen 1 und 2. $=>$ 3 NOPs eingespart

Pipelining

3. Minimieren Sie die Anzahl der notwendigen `NOP`s durch Veränderung der Codereihenfolge, ohne die Funktionalität des Codes zu beeinflussen.

```
1  add   $t1, $t2, $t3
2  sub   $t4, $t2, $t5
3  or    $t6, $t1, $t4
4  and   $t7, $t8, $t9
5  lw    $t9, 4($t7)
6  sub   $t2, $t1, $t3
7  lw    $t1, 16($t9)
```

```
and   $t7, $t8, $t9
add   $t1, $t2, $t3
sub   $t4, $t2, $t5
lw    $t9, 4($t7)
sub   $t2, $t1, $t3
or    $t6, $t1, $t4
lw    $t1, 16($t9)
```

- Durch Verschieben von Zeile 3 soweit wie möglich ans Ende, wird der Konflikt zwischen Zeile 2 und 3 aufgelöst. Außerdem ersetzt der Befehl jetzt das NOP zwischen Zeile 6 und 7. => 2 NOP eingespart

Pipelining

4. Wie groß ist der CPI des Code-Abschnitts vor und nach Ihrer Optimierung? Gehen Sie davon aus, dass die Pipeline zu Beginn der Code-Ausführung leer ist. Welchen Speed-Up konnten Sie erzielen?

```
1  add   $t1, $t2, $t3
2  sub   $t4, $t2, $t5
3  or    $t6, $t1, $t4
4  and   $t7, $t8, $t9
5  lw    $t9, 4($t7)
6  sub   $t2, $t1, $t3
7  lw    $t1, 16($t9)
```

   - Es dauert 5 Takte, bis der erste Befehl abgearbeitet ist (Pipeline leer). Danach beträgt der CPI $= 1$.
   - Es gilt also: $\#Takte = 5 + (\#Instr - 1 + \#NOP)$ und $CPI = \frac{\#Takte}{\#Instr}$
   - $CPI_{alt} = \frac{5+(7-1+5)}{7} = \frac{16}{7}$
   - $CPI_{neu} = \frac{5+(7-1+0)}{7} = \frac{11}{7}$
   - $S = \frac{CPI_{alt}}{CPI_{neu}} = \frac{16}{11} = 1,46$

Caches

Jeder Rechenkern des neuen IBM Power8 Prozessors verfügt über einen 64KB großen, 8-fach satzassoziativen primären (= Level1) Datencache mit 128B großen Blöcken. Die Power8 Architektur arbeitet mit 64bit großen Adressen.

1. Wie lang ist der L1-Cache Index und wie lang ist der Tag? Geben Sie die Größen in bit an.
2. Auf welchen Satz wird die hexadezimale Byte-Adresse

   `0x0123 4567 89ab cdef`

   abgebildet? Geben Sie den Index als Dualzahl, Hexadezimalzahl oder Dezimalzahl an.

Caches

1. Wie lang ist der L1-Cache Index und wie lang ist der Tag? Geben Sie die Größen in bit an.

- Speicher ist Byte-adressierbar, d.h. es muss zwischen $128B = 2^7 B$ unterschieden werden können.

$$Offset = ld(Blockgrösse)bit = ld(128)bit = 7bit$$

- Berechnung der Anzahl an Sätzen mittels Formel zur Kapazität: $Kapazität = \#Sätze \cdot Assoziativität \cdot Blockgrösse$

$$64KB = \#Sätze \cdot 8 \cdot 128B$$

$$\#Sätze = \frac{64 \cdot 1024B}{8 \cdot 128B} = \frac{8 \cdot 2^{10}}{2^7} = 2^{3+10-7} = 2^6$$

Also werden für den Index $ld(2^6)bit = 6bit$ benötigt.

- $\#Adresse = \#Tag + \#Index + \#Offset$

$$64bit = \#Tag + 6bit + 7bit$$

$$\#Tag = 64bit - 13bit = 51bit$$

Caches

2. Auf welchen Satz wird die hexadezimale Byte-Adresse

   `0x0123 4567 89ab cdef`

   abgebildet? Geben Sie den Index als Dualzahl, Hexadezimalzahl oder Dezimalzahl an.

`0x0123 4567 89ab cdef` = {Tag, Index, Offset}
Es müssen Bits 8 - 13 betrachtet werden, da die oberen 51bit den Tag und die unteren 7bit den Offset bilden:
`0x...cdef =...  1100 1101 1110 1111`
Die Adresse wird auf Index $011011_2 = 27_{10} = 0x1B_{16}$ abgebildet.

Caches

Insgesamt verfügt der Prozessor über vier Cache-Ebenen. Der Einfachheit halber betrachten wir aber nur die ersten beiden (L1 und L2 Cache) und zählen L3 und L4 Cache zum Hauptspeicher; ihre Zugriffszeiten müssen also nicht gesondert betrachtet werden. Nehmen wir an, die verschiedenen Speicher verfügen über die folgenden Zugriffszeiten:

Level 1 Datencache	3 Taktzyklen
Level 2 Datencache	12 Taktzyklen
Hauptspeicher	130 Taktzyklen

Eine Messung auf dem System hat außerdem folgende Fehlzugriffsraten für unser Programm ergeben:

Fehlzugriffsrate Level 1 Cache	2%
Fehlzugriffsrate Level 2 Cache	20%

3. Erklären Sie die hohe Fehlzugriffsrate im Level 2 Cache.
4. Berechnen Sie die durchschnittliche Speicherzugriffszeit (*average memory access time, AMAT*) für unser Programm.

Caches

Eine Messung auf dem System hat außerdem folgende Fehlzugriffsraten für unser Programm ergeben:

Fehlzugriffsrate Level 1 Cache	2%
Fehlzugriffsrate Level 2 Cache	20%

3. Erklären Sie die hohe Fehlzugriffsrate im Level 2 Cache.

Der L1-Cache hat eine sehr geringe Miss-Rate und nutzt daher die räumliche und temporale Lokalität des Progammes sehr gut aus. Wenn Daten nun nicht im L1-Cache zu finden sind, ist die Wahrscheinlichkeit hoch, dass die Daten aufgrund mangelnder Lokalität auch nicht im L2-Cache zu finden sind.

Caches

Level 1 Datencache	3 Taktzyklen
Level 2 Datencache	12 Taktzyklen
Hauptspeicher	130 Taktzyklen
Fehlzugriffsrate Level 1 Cache	2%
Fehlzugriffsrate Level 2 Cache	20%

4. Berechnen Sie die durchschnittliche Speicherzugriffszeit (*average memory access time, AMAT*) für unser Programm.

Formel für AMAT: $AMAT = HitTime + MissRate \cdot MissPenalty$
AMAT muss nun für den L1- und den L2-Cache betrachtet werden:

$$AMAT_{Gesamt} = HitTime_{L1} + MissRate_{L1} \cdot AMAT_{L2}$$

mit $AMAT_{L2} = HitTime_{L2} + MissRate_{L2} \cdot HitTime_{Hauptspeicher}$
Durch Einsetzen in die Formel erhält man
$AMAT_{Gesamt} = 3cc + 0,02 \cdot (12cc + 0,2 \cdot 130cc)$

$$= 3cc + 0,02(12cc + 26cc) = 3cc + 0,76cc = 3,76cc$$

Alternativ: $HitRate \cdot HitTime + MissRate \cdot MissPenalty = 3,652cc$

Leistung

Ein Notebooks hat einen Energiesparmodus und einen Performancemodus. Im Hochleistungsbetrieb läuft der Prozessor bei einer Frequenz von 3 GHz und hat einen durchschnittlichen Durchsatz von $1,5 \cdot 10^9$ Befehlen pro Sekunde. Dabei verbraucht der Prozessor 95 Watt. Im Energiesparmodus wird die Frequenz auf 2,3 GHz gesenkt und der Prozessor verbraucht nur noch 35 Watt (zur Erinnerung: $1W = 1\frac{J}{s}$). Der Prozessor verbringt durchschnittlich 20% seiner Zeit mit der Ausführung des Betriebssystems.

1. Wie hoch ist der Durchsatz an Instruktionen für ein Anwendungsprogramm, wenn dieses zusätzlich zum Betriebssystem auf dem Prozessor ausgeführt wird? Geben Sie den Wert sowohl für den Hochleistungs - als auch den Energiesparmodus an.

Leistung

Aufgrund des Betriebssystems stehen nur 80% des maximalen Durchsatzes für Anwendungsprogramme zur Verfügung. Das bedeutet für den Hochleistungsbetrieb:

$$IPS_{hoch} = 0,8 \cdot IPS_{max}$$

$$= 0,8 \cdot 1,5 \cdot 10^9 \frac{Instruktionen}{s} = 1,2 \cdot 10^9 \frac{Instruktionen}{s}$$

Im Energiesparbetrieb verringert sich der Durchsatz linear mit der Frequenz, da für die Abarbeitung des gleichen Programms zwar genausoviele Takte benötigt werden (CPI ist konstant), jeder Takt aber länger dauert. Es ergibt sich

$$IPS_{spar} = \frac{f_{neu}}{f_{alt}} \cdot IPS_{hoch}$$

$$= \frac{2,3GHz}{3GHz} \cdot 1,2 \cdot 10^9 \frac{Instruktionen}{s} = 0,92 \cdot 10^9 \frac{Instruktionen}{s}$$

Leistung

Der Nutzer möchte nun auf dem Notebook ein Anwendungsprogramm mit $5 \cdot 10^9$ Befehlen ausführen.

2. Wie lang sind die Ausführungszeiten im Hochleistungs- und im Energiesparmodus?
3. Wie hoch ist die prozentuale Energieeinsparung, wenn das Programm im Energiesparmodus ausgeführt wird?
4. Wie verändert sich die Ausführungszeit des Programms, wenn nach 2s in den Akkubetrieb gewechselt wird?

Leistung

Der Nutzer möchte nun auf dem Notebook ein Anwendungsprogramm mit $5 \cdot 10^9$ Befehlen ausführen.

2. Wie lang sind die Ausführungszeiten im Hochleistungs- und im Energiesparmodus?

Es gilt: $T = \frac{N_{Instruktionen}}{Durchsatz}$

$$T_{hoch} = \frac{5 \cdot \cancel{10^9} Instruktionen}{1,2 \cdot \cancel{10^9} \frac{Instruktionen}{s}} = 5 \cdot \frac{5}{6} Instruktionen \cdot \frac{s}{Instruktionen}$$

$$= 4\frac{1}{6}s \approx 4,17s$$

Analog für $T_{spar} = \frac{5}{0,92}s \approx 5,435s$

Leistung

3. Wie hoch ist die prozentuale Energieeinsparung, wenn das Programm im Energiesparmodus ausgeführt wird?

Aus der Aufgabenstellung ist bekannt, dass $1W = 1\frac{J}{s}$. Bekannt sind Leistungsverbrauch und Ausführungsdauer der Programme. Es ergibt sich:

$$Energie = Leistung \cdot Ausführungszeit$$

Für den jeweiligen Betriebsmodus ausgerechnet:

$$E_{hoch} = 95W \cdot 4,17s = 396,15J$$

$$E_{spar} = 35W \cdot 5,435s = 190,225J$$

Eingespart wird daher

$$E_{diff} = 396,15J - 190,225J = 205,925J$$

bzw.

$$\frac{205,925}{396,15} = 0,5198 = 51,98\%$$

Leistung

4. Wie verändert sich die Ausführungszeit des Programms, wenn nach 2s in den Akkubetrieb gewechselt wird?

Nach den zwei Sekunden im Hochleistungsbetrieb verbleiben noch

$$5 \cdot 10^9 \mathit{Instruktionen} - 2s \cdot 1,2 \cdot 10^9 \frac{\mathit{Instruktionen}}{s}$$

$$= (5 - 2,4) \cdot 10^9 \mathit{Instruktionen}$$

Diese werden in

$$\frac{2,6 \cdot \cancel{10^9} \mathit{Instruktionen}}{0,92 \cdot \cancel{10^9} \frac{\mathit{Instruktionen}}{s}} = 2,826s$$

ausgeführt. Es ergibt sich eine neue Ausführungszeit von

$$T_{neu} = 2s + 2,826s = 4,826s$$