Aufgabe 1: CPU MIPS Ausführungszeit

cpumemcpy.tex

Der RP2040 Chip auf dem Raspberry PI Pico Board enthält einen ARM Cortex M0+ Prozessor. Nehmen Sie nun an, dass der Prozessor den MIPS Befehlssatz ausführen kann - eine fiktionale Annahme, denn in der Realität unterstützt der Prozessor den ARM Befehlssatz. Die Funktion „memcpy“kopiert „num“Elemente aus einem Array „src“in ein Array „dest“. In Listing 1 ist der Code in Rust und in Listing 2 ist der Code in C angegeben. Listing 3 zeigt den MIPS Assemblercode ohne die Checks im Rustcode in den Zeilen 2 und 3. Die Argumente der memcpy Funktion werden in den Prozessorregistern gemäß Tabelle 1 übergeben.

```
1 pub fn memcpy(num: usize, src: &[i32], dest: &mut [i32]) {
2     if num >= src.len() { return };
3     if num >= dest.len() { return };
4     for i in 0..num {
5         dest[i] = src[i];
6     }
7 }
```

Listing 1: memcpy in Rust

```
void memcpy(unsigned int num, int src[], int dest[]) {
    int i = 0;
    while (i < num) {
        dest[i] = src[i];
        i = i + 1;
    }
}
```

Listing 2: memcpy in C

Tabelle 1: Registerinhalte beim Funktionsaufruf von memcpy

Register	Parameter
$4	num
$5	Zeiger auf das erste Element des Arrays src
$6	Zeiger auf das erste Element des Arrays dest

```
memcpy(unsigned int, int*, int*):
        beq     $4,$0,$L9
        sll     $4,$4,2
        addu    $4,$5,$4
$L3:
        lw      $2,0($5)
        sw      $2,0($6)
        addiu   $5,$5,4
        addiu   $6,$6,4
        bne     $5,$4,$L3
        nop
$L9:
        jr      $31
        nop
```

Listing 3: memcpy in MIPS Assembler

Schätzen Sie die Ausführungszeit für einen Aufruf von „memcpy“mit den folgenden Parametern memcpy(1000, 0x20010000, 0x20020000) ab. Geben sie in ihrer Antwort ausführlich ihre Annahmen an.

Abschätzung der Ausführungszeit für `memcpy` auf dem RP2040

Um die Ausführungszeit für den Aufruf der `memcpy`-Funktion im MIPS-Assembler abzuschätzen, wenn der Parameter `num` den Wert 1000 hat, müssen wir die Anzahl der ausgeführten Befehle und deren Zyklen pro Befehl (CPI) bestimmen. Wir gehen davon aus, dass der Prozessor des RP2040 den MIPS Befehlssatz ausführen kann.

Gegebener MIPS-Assemblercode (Auszug aus Listing 3)

```
memcpy (unsigned int, int*, int*):
beq $4,$0,$L9     # Branch if num == 0
sll $4,$4,2       # num = num * 4 (shift left by 2 bits for byte offset)
addu $4,$5,$4     # $4 = src + (num * 4) -> $4 holds end address of src
$L3:              # Loop start
lw $2,0($5)       # Load word from src[i] into $2
sw $2,0($6)       # Store word from $2 into dest[i]
addiu $5,$5,4     # Increment src pointer by 4 bytes
addiu $6,$6,4     # Increment dest pointer by 4 bytes
bne $5, $4,$L3    # Branch if src pointer != end address ($4)
nop               # Branch delay slot
$L9:              # Loop end / exit
jr $31            # Jump to return address
nop               # Branch delay slot
```

Registerbelegung (gemäß Tabelle 1)

- `$4`: `num` (Anzahl der zu kopierenden Elemente)
- `$5`: Zeiger auf das erste Element des Arrays `src`
- `$6`: Zeiger auf das erste Element des Arrays `dest`

Annahmen für die Abschätzung

a) **CPU-Taktfrequenz:** Der ARM Cortex-M0+ Prozessor auf dem RP2040 Chip kann standardmäßig mit bis zu **133 MHz** getaktet werden. Wir verwenden diesen Wert.

$$\text{Taktfrequenz} = 133\,\text{MHz} = 133 \times 10^6\,\text{Hz}$$

b) **CPI (Cycles Per Instruction):** Wir gehen von einem **idealen Pipelining** aus, bei dem jeder Befehl durchschnittlich **1 Taktzyklus (CPI = 1)** benötigt. Dies ist eine Vereinfachung. In der Realität können Befehle (insbesondere Ladebefehle wie `lw`) Pipeline-Stalls verursachen, und die tatsächlichen CPI-Werte können variieren.

c) **Speicherzugriff:** Wir nehmen an, dass alle Speicherzugriffe (für `lw` und `sw`) einen Taktzyklus benötigen.

d) **Parameter:** `num = 1000`.

Analyse der Befehlsausführung

Die Ausführung des Codes kann in drei Phasen unterteilt werden: Initialisierung, Schleifendurchläufe und Abschluss.

I. Initialisierungsphase (vor der Schleife)

Diese Befehle werden einmalig ausgeführt.

a) `beq $4,$0,$L9`: Wird ausgeführt. Da `num` (in `$4`) 1000 ist, ist es nicht gleich 0, also wird nicht gesprungen.

b) `sll $4,$4,2`: Wird ausgeführt. `$4` wird zu `1000 * 4 = 4000`. Dies konvertiert die Elementanzahl in eine Byte-Offset-Adresse, da jedes Element 4 Byte (ein Wort) groß ist.

c) `addu $4,$5,$4`: Wird ausgeführt. `$4` enthält nun die Endadresse des `src`-Arrays (`start_src_address + 4000`). Dieser Wert dient als Abbruchkriterium für die Schleife.

Anzahl Befehle Initialisierung: 3 Befehle.
Ausführungszeit Initialisierung: 3×1 Taktzyklus/Befehl $= 3$ Taktzyklen.

II. Schleifenphase (`$L3` bis `bne`)

Diese Schleife wird `num` mal durchlaufen. Da `num = 1000`, wird die Schleife **1000 Mal** ausgeführt. In jedem Schleifendurchlauf werden folgende Befehle ausgeführt:

a) `lw $2,0($5)`: (Load Word) Lädt ein Element aus dem `src`-Array.

b) `sw $2,0($6)`: (Store Word) Speichert das Element in das `dest`-Array.

c) `addiu $5,$5,4`: Inkrementiert den `src`-Zeiger um 4 Bytes (zum nächsten Element).

d) `addiu $6,$6,4`: Inkrementiert den `dest`-Zeiger um 4 Bytes (zum nächsten Element).

e) `bne $5, $4,$L3`: (Branch if not equal) Vergleicht den aktuellen `src`-Zeiger mit der berechneten Endadresse. Wenn nicht gleich, wird zurück zum Schleifenanfang gesprungen.

f) `nop`: (No Operation) Dieser Befehl füllt den Branch-Delay-Slot des `bne`-Befehls und wird immer ausgeführt, auch wenn der Sprung nicht genommen wird.

Anzahl Befehle pro Schleifendurchlauf: 6 Befehle.
Ausführungszeit pro Schleifendurchlauf (angenommener CPI=1 für alle): 6×1 Taktzyklus/Befehl $=$ 6 Taktzyklen.
Gesamtausführungszeit Schleife: 1000 Durchläufe$\times 6$ Taktzyklen/Durchlauf $= 6000$ Taktzyklen.

III. Abschlussphase (nach der Schleife)

Diese Befehle werden einmal ausgeführt, nachdem die Schleife beendet ist (wenn `num` ursprünglich 0 war oder nach dem letzten Schleifendurchlauf).

a) `jr $31`: (Jump Register) Kehrt zur aufrufenden Funktion zurück.

b) `nop`: (No Operation) Füllt den Branch-Delay-Slot des `jr`-Befehls.

Anzahl Befehle Abschluss: 2 Befehle.
Ausführungszeit Abschluss: 2×1 Taktzyklus/Befehl $= 2$ Taktzyklen.

Gesamtzahl der Taktzyklen

Die Gesamtzahl der Taktzyklen für den gesamten Funktionsaufruf ist die Summe der Zyklen aus allen Phasen:

$$\text{Gesamt-Taktzyklen} = 3\,(\text{Init}) + 6000\,(\text{Schleife}) + 2\,(\text{Abschluss}) = \textbf{6005 Taktzyklen}$$

Berechnung der Ausführungszeit

Mit der angenommenen Taktfrequenz von 133 MHz:

$$\text{Ausführungszeit} = \frac{\text{Gesamt-Taktzyklen}}{\text{Taktfrequenz}}$$

$$\text{Ausführungszeit} = \frac{6005\ \text{Taktzyklen}}{133 \times 10^6\ \text{Taktzyklen/Sekunde}}$$

$$\text{Ausführungszeit} \approx 45.149 \times 10^{-6}\ \text{Sekunden}$$

$$\text{Ausführungszeit} \approx \textbf{45.149 Mikrosekunden (µs)}$$

Zusammenfassung und wichtige Hinweise

Unter den genannten Annahmen (RP2040-Taktfrequenz von 133 MHz, idealisiertes Pipelining mit CPI=1 und Speicherzugriff innerhalb eines Taktes) beträgt die geschätzte Ausführungszeit für den `memcpy`-Aufruf mit `num = 1000` Elementen ungefähr **45.149 Mikrosekunden**.

Es ist zu beachten, dass diese Schätzung idealisiert ist. In der Praxis können Faktoren wie tatsächliche Pipeline-Stalls (insbesondere durch Lade-/Speicherbefehle), Branch-Prediction-Fehler und die spezifische Implementierung der MIPS-Architektur auf dem ARM Cortex-M0+ (falls es eine Emulation oder Transkompilierung gäbe) die tatsächliche Ausführungszeit beeinflussen. Diese Faktoren würden die Ausführungszeit in der Regel erhöhen.

Antwort KI generiert mit Gemini 2.5 Flash am 29.6.2025 (angepasst)

Aufgabe 2: Ausführungszeit vs. Cache Hit Rate

cpuhitperf.tex

Ein Rechnersystem ist mit einem 2kB großen L1 Datencache, einem 4kB großen L1 Befehlscache und einem DRAM mit einer Größe von 16 Gigabyte aufgebaut. Ein Zugriff auf den L1 Cache dauert 2 Taktzyklen. Ein Zugriff auf das DRAM dauert 300 Taktzyklen. Sie haben ein Programm zur Positionsschätzung aus LiDAR Daten geschrieben. Sie haben ein Codeprofiling mit Testdaten durchgeführt. Dabei haben sie die Cachezugriffsdaten aus Tabelle 2 gemessen. Zusätzlich haben sie mit dem Profiling herausgefunden, dass 15 Prozent aller Instruktionen load oder store Befehle sind. Nehmen sie eine einfache MIPS Prozessorarchitektur an. Bei einem Cachehit dauert die Ausführung eines Befehls inklusive des Speicherzugriffs im Durchschnitt 2 Takte.

Tabelle 2: Zugriffsfehler auf den Instruction- Datacache

Cache	Missrate
Instruktion	0.4%
Data	3.7%

Schätzen Sie ab wie sich die Ausführungszeit ihres Programm im Vergleich zu einem idealen Cacheverhalten verhält.

Ideales Verhalten

Bei einem idealen Cacheverhalten sind alle Befehls- und alle Datenzugriffe Treffer im Cache. Man hat also eine Cache Hit Rate von 100 Prozent. Im Durchschnitt dauert die Bearbeitung eines Befehls dann zwei Taktzyklen. Damit ergibt sich ein idealer CPI (Cycles per Instruction) Wert von CPI $=$ 2.

Verhalten aufgrund von Zugriffsfehlern auf den Befehlscache

Bei einem Instructionfetch auf einen Befehl, der nicht im Cache ist, muss der Befehl aus dem DRAM geholt werden. In diesem Fall dauert es 298 Takte länger bis der Befehl in der CPU ist. Das tritt in 0.4 Prozent aller Zugriffe auf den Befehlsspeicher ein. In Bezug auf die durchschnittlich benötigte Anzahl von Taktzyklen für die Ausführung eines Befehls aufgrund der Fehlzugriffe auf den Befehlscache ergibt sich

$$\begin{aligned} CPI_{act,B} &= CPI_{ideal} + \text{Missrate,B} \times \text{Misspenalty,DRAM} \\ &= 2 + 0.004 \times 298 = 2 + 1.192 = 3.192 \end{aligned}$$

Nur aufgrund der Fehlzugriffe auf den Befehlscache dauert die Ausführung eines Befehls 3.192 Taktzyklen statt 2 Taktzyklen. Die Ausführungszeit wird damit etwa 60 Prozent größer.

Verhalten aufgrund von Zugriffsfehlern auf den Datencache

15 Prozent aller Befehle sind load oder store Befehle, die einen Zugriff auf den Datencache beinhalten. Wenn die Daten bei dem Zugriff nicht im Cache sind, dann dauert die Ausführung des Befehls 298 Takte länger. Das tritt in 3.7 Prozent aller Datenzugriffe ein. Damit ergibt sich für die durchschnittlich benötigte Anzahl von Taktzyklen für die Ausführung eines Befehls aufgrund von Wartezeiten bei Datenzugriffen:

$$\begin{aligned} CPI_{act,D} &= CPI_{ideal} + \text{Anteil Befehle mit Datenzugriff} \times \text{Missrate,D} \times \text{Misspenalty,DRAM} \\ &= 2 + 0.15 \times 0.037 \times 298 \\ &= 2 + 1.654 = 3.654 \end{aligned}$$

Aufgrund von Fehlzugriffen auf den Datencache erhöht sich die durchschnittlich benötigte Anzahl von Takten für die Ausführung eines Befehls von 2 auf 3,654 Takte. Dies entspricht einer Erhöhung der Ausführungszeit um 83 Prozent.

Verhalten mit Auswirkungen von Befehls- und Datencache

Die Fehlzugriffe auf Befehls- und Datencache wirken unabhängig voneinander. Die Anzahl von benötigten Taktzyklen pro Befehl ist dann

$$\begin{aligned} CPI_{act} &= CPI_{ideal} + \text{Zusatzzyklen Fehlzugriffe Befehlscache} + \text{Zusatzzyklen Fehlzugriffe Datencache} \\ &= 2 + 1.192 + 1.654 = 2 + 2.846 = 4.846 \end{aligned}$$

Es dauert im Durchschnitt also 4,846 Taktzyklen statt 2 Taktzyklen um einen Befehl auszuführen wenn die Zugriffsfehler auf den Befehls- und Datencache berücksichtigt werden. Die Ausführungszeit steigt auf das 2,4 fache der Ausführungszeit mit einem idealen Cache. Das entspricht einer Steigerung der benötigten Ausführungszeit für das Programm um 142 Prozent.

Aufgabe 3: Entwicklung der CPU Technologie

50yproc.tex

In Abbildung 1 ist der zeitliche Verlauf von Technologieparametern der Mikroprozessortechnologie über die letzten 50 Jahre dargestellt. In der Abbildung fehlt die Legende.

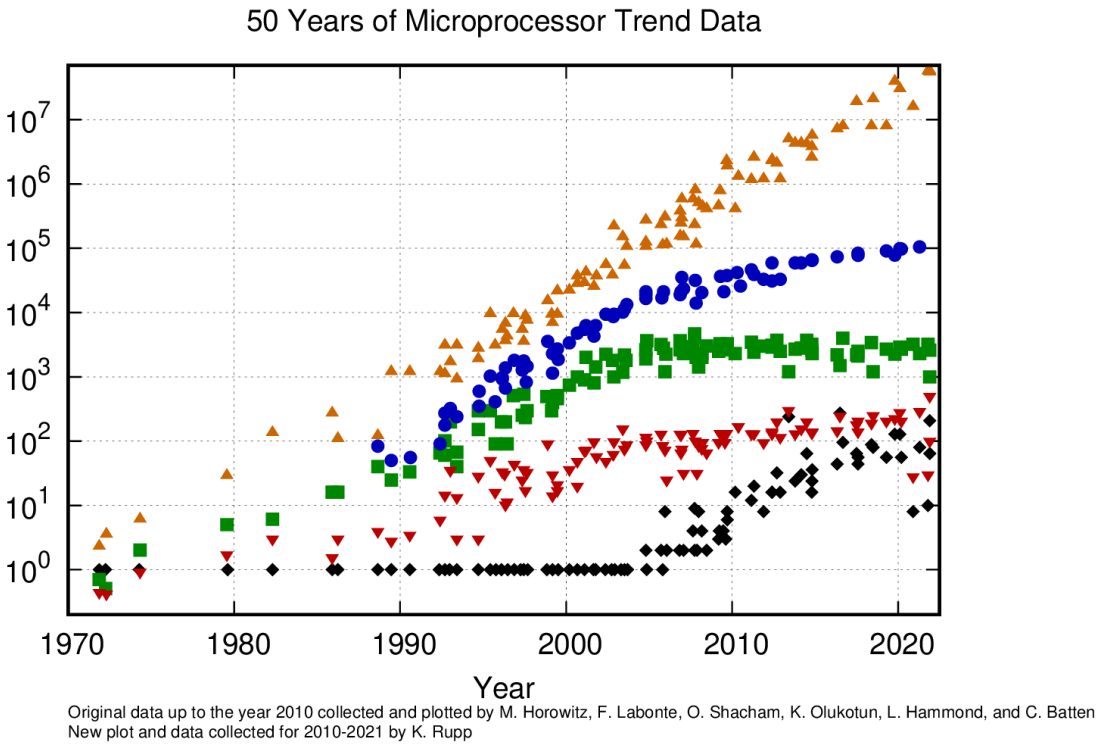

Abbildung 1: Entwicklung der Prozessortechnologie

Ergänzen sie die Legende. Ordnen sie die Begriffe

- Number of Logical Cores
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Transistors (thousands)
- Typical Power (Watts)

den Datenpunkten zu.

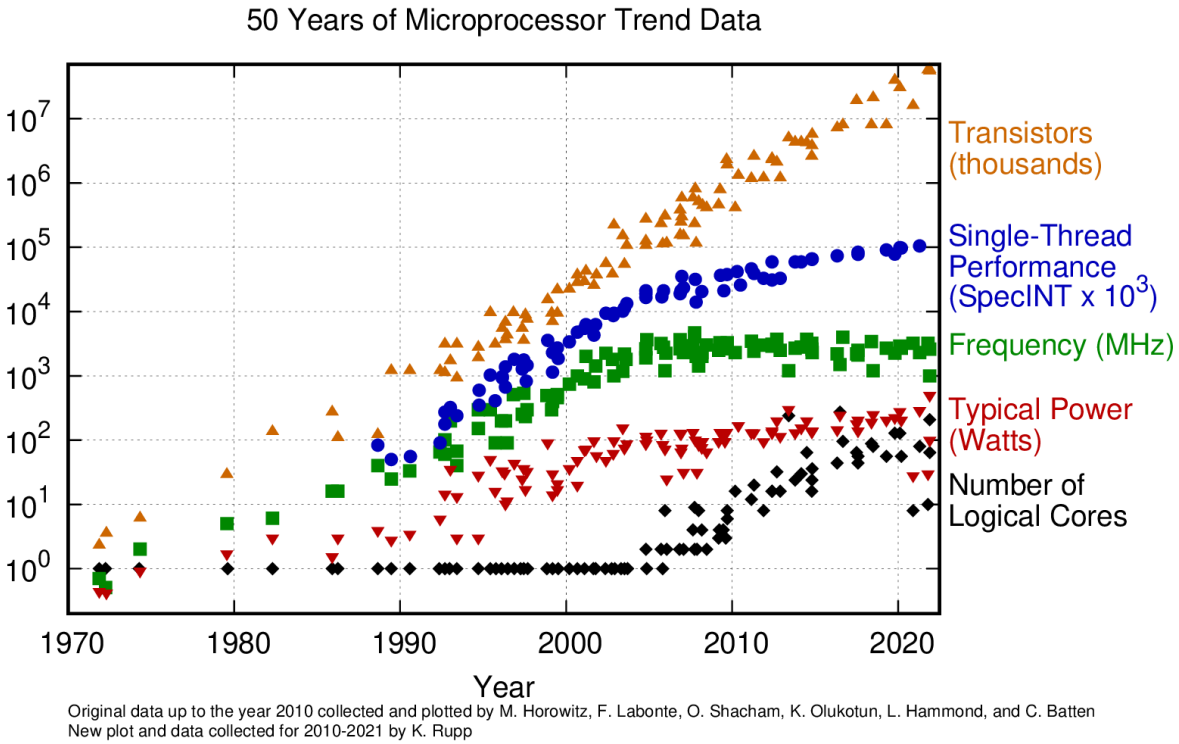

Abbildung 2: Entwicklung der Prozessortechnologie - mit Legende

Karl Rupp, 42 years of Microprocessor Trend data, https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/