6.1 Teil A: Maschinencode (Hex) → Assembler

1. 0x012A5820
   0000 0001 0010 1010 0101 1000 0010 0000
   OpCode = 000000 = R-Format (Befehle die nur mit Registern arbeiten)
   X-Bus-Adresse / rs / Source Register / = 01 001 = 9
   Y-Bus-Adresse / rt / Target Register / = 0 1010 = 10
   Z-Bus-Adresse / rd / Destination Register / = 0101 1 = 11
   shamt / Shift Amount = 00000 (dieser Parameter wird hier nicht gebraucht)
   Funktionscode = 100000 (steht für add)
   => add $t1, $t2, $t3 (Registernummer 8 – 15 sind die temp Register $t0 bis $t7)
2. 0x212A0004
   001000|01001|01010|0000000000000100
   OpCode = 001000 = OpCode für add immidiate (I-Format-Befehl)
   X-Bus-Adresse / rs / Source Register / = 01001 = 9
   Y-Bus-Adresse / rt / Target Register / = 01010 = 10
   imm. = 0000000000000100 = 4
   => addi $ $t1, $t2, 4
3. 0x112A0003
   0001 00 | 01 001 | 0 1010 | 0000 0000 0000 0011
   OpCode = 000100 = beq (branch if equal) (I-Format-Befehl)
   rs = 01001 = Register 9 = $t1
   rt = 01010 = Register 10 = $t2
   `immediate = 0000000000000011` = 3
4. 0x0800000C
   0000 10 | 00 0000 0000 0000 0000 0000 1100
   OpCode = 000010 = jump (J-Format-Befehl)
   Target Adresse = 00 0000 0000 0000 0000 1100
   Target Adresse um 2 nach links geshiftet = 00 0000 0000 0000 0011 0000 = j 0x000030
   => Sprung zur Adresse 0x000030 bzw. in der Adresse steht das Label der Funktion oder loop.

6.2 Teil B: Assembler → Maschinencode (Hex)

1. add $t1, $t2, $t3 = 0000 0001 0010 1010 0101 1000 0010 0000
2. addi $s1, $s2, 8 = 001000 | 10001 | 10010 | 0000000000001000
   $s1 = 17 = 10001 $s2 = 18 = 10010 8 = 000000001000
   Hex = 0010 0010 0011 0010 0000 0000 0000 1000 = 0x22320008
3. beq $t0, $t1, 8 = 000100 | 01000 | 01001 | `0000000000001000`
   `Hex = 0x11090008`
4. `j 0x00000020` => 000010 | 00 0000 0000 0000 0010 0000
   2 shifts nach rechts => 000010 | 00 0000 0000 0000 0000 1000 => 0x0800008

6.4 RISC, CISC und SIMD

a) RISC (Reduced Instruction Set Computer)

- Befehlslänge: Fest (typisch 32 Bit, z. B. MIPS, ARM).
- Befehlskomplexität: Einfache, einheitliche Befehle (z. B. nur LOAD, STORE, ADD).
- Anzahl der Register: Viele (16–32 General-Purpose-Register).
- Anwendungsbereiche: Eingebettete Systeme (Mikrocontroller), Mobile Geräte (ARM), Hochleistungsrechner (RISC-V, PowerPC).

b) CISC (Complex Instruction Set Computer)

- Befehlslänge: Variabel (z. B. Intel x86: 1–15 Byte).
- Befehlskomplexität: Komplexe Befehle (z. B. MOVSB, AAM – eine Instruktion kann viel tun).
- Anzahl der Register: Weniger (z. B. x86: 8–16 General-Purpose-Register).
- Anwendungsbereiche: Desktop-PCs (Intel/AMD x86), Legacy-Systeme.

c) SIMD (Single Instruction, Multiple Data)

- Befehlslänge: Variabel, oft als Erweiterung zu RISC/CISC (z. B. SSE, AVX, NEON).
- Befehlskomplexität: Spezialisiert auf parallele Datenverarbeitung (z. B. ADD auf 4 Floats gleichzeitig).
- Anzahl der Register: Spezielle Vektorregister (z. B. 128–512 Bit breit).
- Anwendungsbereiche: Multimedia (Videoencoding), Wissenschaftliche Berechnungen, KI (Matrixoperationen).

- **Intel x86: CISC (variabel lange Befehle, komplexe Instruktionen).**
- **ARM Cortex: RISC (feste 32/64-Bit-Befehle, energieeffizient).**
- **GPUs mit CUDA: SIMD (massiv parallele Verarbeitung, z. B. NVIDIA-GPUs).**