Karlsruher Institut für Technologie
Institut für Technische Informatik (ITEC)

Rechnerorganisation im WS 2020/21

Musterlösungen zum 5. Übungsblatt

Prof. Dr. Jörg Henkel
Dr.-Ing. Lars Bauer
Roman Lehmann, M. Sc.
Haid-und-Neu-Str. 7,
Geb. 07.21 (Technologiefabrik)
Email: roman.lehmann@kit.edu

Lösung 1 (4 Punkte)

1. Bei einem Pseudobefehl handelt es sich um ein Mnemonik, das vom Assemblierer nicht auf einen Maschinenbefehl abgebildet wird. Es findet eine Ersetzung durch einen oder mehrere andere Befehle statt. Der Befehlsumfang kann mit dieser Technik für den Programmierer einfach erweitert werden, ohne dass die Komplexität der CPU erhöht wird. 2 P.

   Pseudobefehle sind üblicherweise direkte Spezialfälle allgemeinerer Befehle (z.B. `b`, `clear` und `neg`) und Befehle wie `li`, die aus technischen Gründen nicht als ein Befehl codiert werden können. Das Problem des `li` Befehls liegt in der festen Breite eines Befehlswortes (32 Bit). Diese feste Länge verhindert das Speichern des Opcodes und der 32 Bit langen Konstanten in einem Befehlswort.

2. In den `$tX`-Registern werden vorwiegend temporäre Variablen abgelegt, während die `$sX`-Register zur längerfristigen Speicherung von Werten verwendet werden. 2 P.

   Entsprechend besagt die Aufrufkonvention, dass der Wert der `$tX`-Register von Funktionen beliebig überschrieben werden darf.

   Vor dem Verändern eines `$sX`-Register muss hingegen der alte Wert gesichert und vor dem Rücksprung wieder hergestellt werden. Das Speichern erfolgt typischerweise auf dem Stack.

   Es handelt sich hierbei um eine Konvention der Programmierer, die keine Auswirkungen auf das Hardware-Design hat. Auf Hardwareebene sind die Register identisch behandelte General Purpose Register.

Lösung 2 (4 Punkte)

	R-Typ	I-Typ	J-Typ	Pseudobefehl
`add`	X			
`j`			X	
`beq`		X		
`nop`				X
`ori`		X		
`subu`	X			
`negu`				X
`lb`		X		X

Hinweis: Bei `lb` mit einer Konstanten, die zu groß ist, um als 16-Bit-Wert dargestellt zu werden, handelt es sich um einen Pseudobefehl (z.B. `lb` $0,0x12345$(`t1`)), ansonsten um einen Befehl vom Typ I. Jede der beiden Lösungsmöglichkeiten wird als korrekt akzeptiert.

Lösung 3 (8 Punkte)

Siehe `blatt05_Rot13_musterloesung.asm`

Lösung 4 (11 Punkte)

1. `MYSTIC CODE` setzt die folgende Switch-Anweisung in MIPS-Assembler um: 2 P.

```
switch ( kobi_dir ) {
    case 0: temp1 = kobi_pos_y - 1;
            break;
    case 1: temp0 = kobi_pos_x + 1;
            break;
    case 2: temp1 = kobi_pos_y + 1;
            break;
    case 3: temp0 = kobi_pos_x - 1;
}
```

   Ein default-Fall wurde nicht implementiert. Auf die Benutzung einer Lookup-Tabelle in Form von je einem Label pro Case wurde hier verzichtet. Stattdessen wird das Sprung-Offset direkt berechnet und die Ausführung an der Programmadresse `Case0` + `kobi_dir` $*$ 8 fortgesetzt.

2. Siehe `blatt05_robot_musterloesung.asm`. 2 P.

3. Siehe `blatt05_robot_musterloesung.asm`. 3 P.

4. Siehe `blatt05_robot_musterloesung.asm`. 4 P.