Aufgabenblatt 7: Adressierung

Einleitung: Die Operanden eines Assembler-Befehls können in unterschiedlichen Formen angegeben werden. Operanden und Werte können beispielsweise **implizit** (auch **inhärent**) im Op-Code des Befehls enthalten sein. So arbeitet die Instruktion *jal* immer auf dem Register *$ra*, ohne dass dies expizit angegeben werden muss. Bei der **unmittelbaren** (**immediate**) Adressierung werden konstante Werte direkt in einem der Operandenfelder der Instruktion angegeben, z.B. *li $t0 42*. Im Weiteren besteht die Möglichkeit **Register**, sowie **Speicher** zu adressieren; dies kann **direkt** oder **indirekt** erfolgen (einstufige oder zweistufige Adressierung). Adressen können **absolut** oder **relativ** (Speicher-relativ oder Register-relativ) berechnet werden. Die **Indexregister-Adressierung** ist eine Form der Speicher-relativen Adressierung, da hier ein Registerinhalt als *Displacement* relativ zu einer in der Instruktion gegebenen Speicheradresse addiert wird. Zu den Register-relativen Adressierungsarten zählt die **Basisregisteradressierung** ggf. **mit Index**. Bei der relativen Adressierung wird die *effektive Adresse* durch eine **Summenbildung** der einzelnen Adressteile gebildet.
Nicht jede Architektur unterstützt alle oben genannten Adressierungsarten für jede Instruktion, insbesondere sind Speicherzugriffe bei Load/Store-Architekturen nur durch load- (lw) und store-Befehle (sw) möglich.

Aufgabe 1: Adressierung in der Praxis

In dieser Aufgabe sollen Sie die aus der Vorlesung bekannten Adressierungsarten für den Pentium und den PowerPC betrachten. Sie sollen dabei lernen, die Designentscheidungen bezüglich Adressierung auf den verschiedenen Architekturen nachzuvollziehen.

a) Aus welchen Adressteilen wird beim Pentium die effektive Adresse berechnet?

b) Welche Adressierungsarten kennt der PowerPC? Wie wird die effektive Adresse bestimmt?

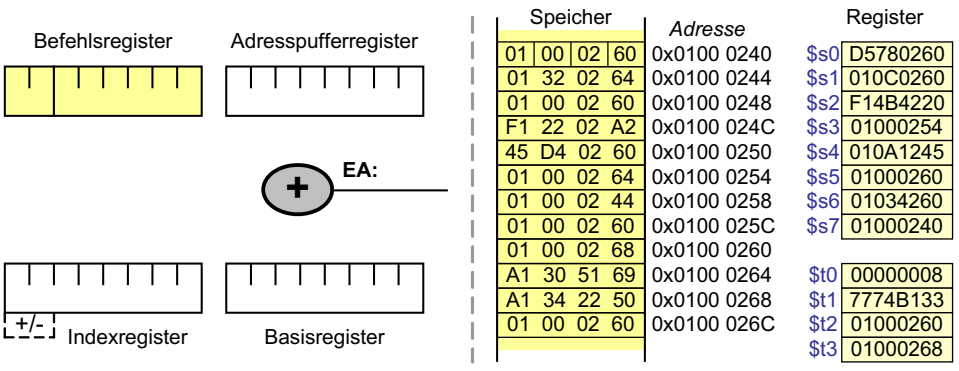
c) Welche Befehle können beim PowerPC den Hauptspeicher adressieren?

d) Vergleichen Sie die beiden Adressierungsarten. Welche Vor- und Nachteile beinhalten die beiden Adressierungsarten und deren Beschränkung auf bestimmte Befehle?

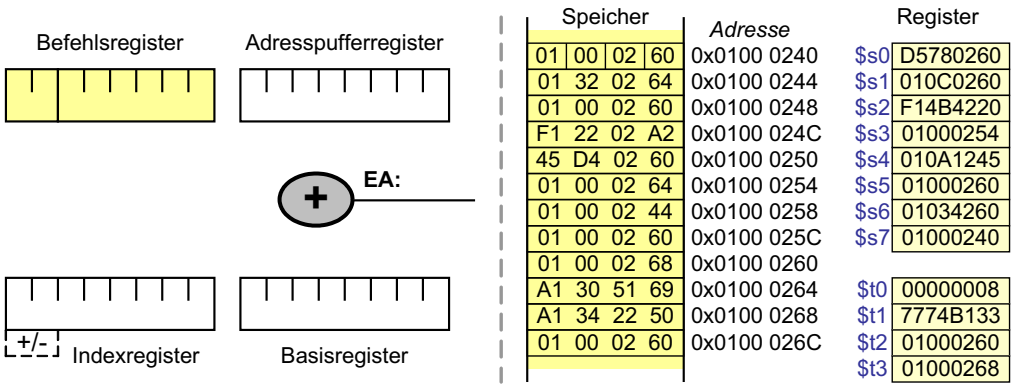
Aufgabe 2: Berechnung der effektiven Adresse

Um welche Form der Adressierung handelt es sich und welche Werte liefern folgende Assembler Befehle? Geben Sie jeweils die effektive Adresse und den dazugehörige Wert an. Zeichnen Sie durch Pfeile in die jeweilige Grafik ein, welche Register zur Berechnung der effektiven Adresse verwendet werden, und welche Werte in den entsprechenden Registern stehen.

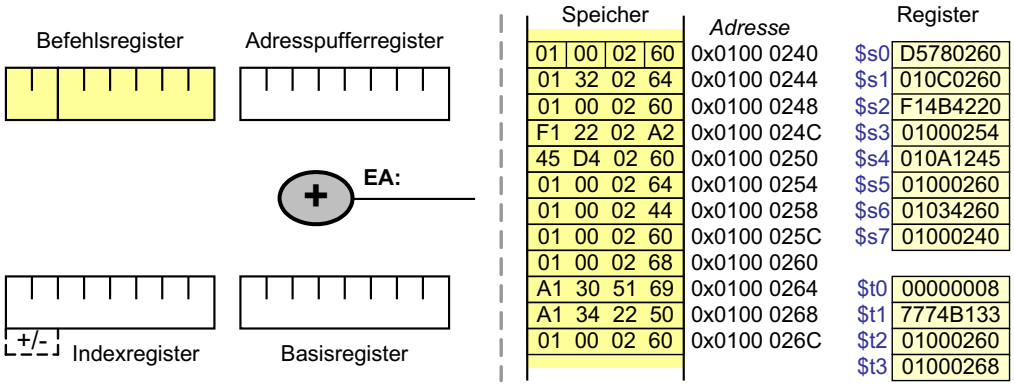
a) move $t0, $s2 Adressierungsart:



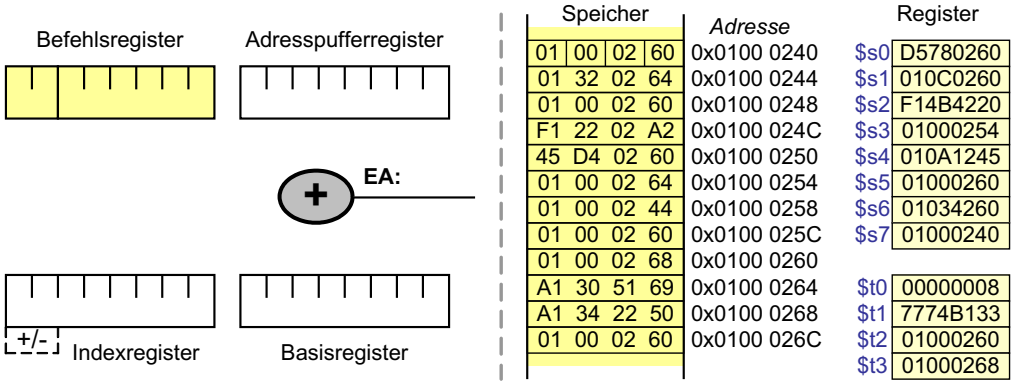
b) li $t2, 123 Adressierungsart:



c) lw $t1 (0x10($s7)($t0)) Adressierungsart:



d) lw $t3 (0x0100024C($t0)) Adressierungsart:



Aufgabe 3: Adressierung am Beispiel von Datentypen

a) Entwickeln Sie Assembler Code, der den String in das (Word-)Array kopiert. Dabei soll auf der linken Seite Code für eine load/store Architektur entwickelt werden, wie sie uns vom MIPS bekannt ist. Schreiben Sie auf der rechten Seite Code für einen CISC Porcessor, der keine load/store Architektur ist. D. h. es dürfen auch Speicherinhalte in arithmetischen Operationen vorkommen.

load / store (MIPS)

```
        .data
        .align 2
txt:    .asciiz „Hallo Welt“
arr:    .space 40
length: .word 0x0A
arrP:   .word arr

        .text
        .globl main
main:
```

CISC

```
        .data
        .align 2
txt:    .asciiz „Hallo Welt“
arr:    .space 40
length: .word 0x0A
arrP:   .word arr

        .text
        .globl main
main:
```

Speicher				*Adresse*
H	A	L	L	0x0100 0240
O	␣	W	E	0x0100 0244
L	T	00	XX	0x0100 0248
XX	XX	XX	XX	0x0100 024C
XX	XX	XX	XX	0x0100 0250
XX	XX	XX	XX	0x0100 0254
XX	XX	XX	XX	0x0100 0258
XX	XX	XX	XX	0x0100 025C
XX	XX	XX	XX	0x0100 0260
XX	XX	XX	XX	0x0100 0264
XX	XX	XX	XX	0x0100 0268
XX	XX	XX	XX	0x0100 026C
00	00	00	0A	0x0100 0270
01	00	02	4C	0x0100 0274
				0x0100 0278