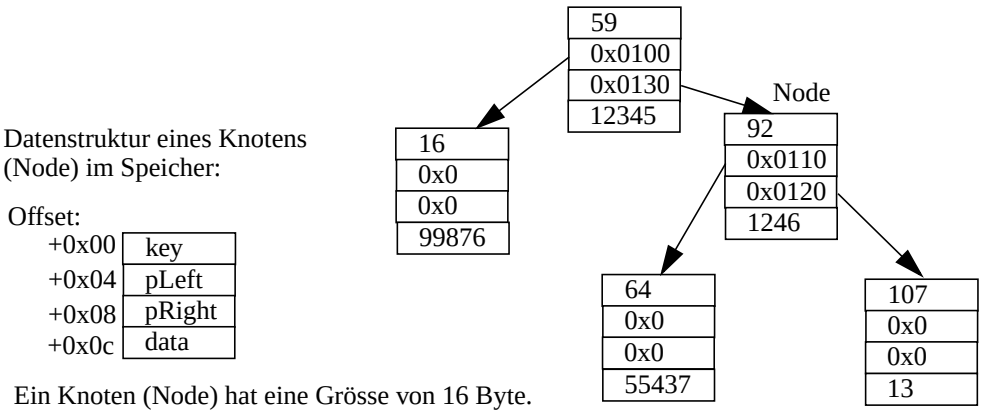
## Aufgabe 1: Assemblerprogrammierung *(Total 20 Punkte)*

Binäre Suchbäume sind oft verwendete Datenstrukturen in der Informatik. Die folgende Abbildung zeigt einen binären Suchbaum:




Der Baum (Tree) ist sortiert nach dem Wert des “key”-Feldes, wobei alle Teilbäume links eines Knotens (Node) kleinere Werte, rechts eines Knotens (Node) grössere Werte aufweisen. Um nun nach einem Knoten (Node) mit vorgegebenem “key” zu suchen, kann folgende Funktion verwendet werden:

```
/* Suchen (lookup) nach “key” im Baum “pTree” */
Node* lookup ( Node* pTree, int key )
{
  while ( pTree != 0 ) {
    if ( pTree-> key == key ) {
      return pTree;
    }
    else if ( pTree->key < key ) {
      pTree = pTree->pLeft;
    }
    else {
      pTree = pTree->pRight;
    }
  }
  return 0;
}
```

*[Hinweis: pTree->pRight adressiert das Feld pRight in der Node-Struktur.]*

**Aufgaben:**

a) *(7 Punkte)*
Programmaufruf: *Schreiben* Sie in MIPS-Assembler einen Aufruf der Lookupfunktion. *Definieren* Sie zuerst, wie die Parameter übergeben werden und das Resultat zurückgegeben wird. Beim Aufruf der Lookupfunktion sind keine temporären Register zu sichern.
*(TIP: Folgen Sie der MIPS-Konvention)*

b) *(13 Punkte)*
*Implementieren* Sie die komplette Lookupfunktion in MIPS-Assembler. Über- und Rückgabe der Parameter sollen wie in Teilaufgabe a) definiert erfolgen.

# Aufgabe 1: Lösung

a)

Definition der Paramter Über- und Rückgabe:
a0 = pTree
a1 = key
v0 = Resultat

jal Lookup

b)

```
Verschiedene Lösungen sind möglich, Beispiel:

loop:
   beq zero, a0, end
   lw t0, 0(a0)
   beq t0, a1, equal  ; (pTree->key == key)
   blt a1, t0, left   ; (pTree->key < key)
   lw a0, 8(a0)
   j loop

left:
   lw a0, 4(a0)
   j loop

equal:
   add v0, a0, zero
   jr ra

end:
   add v0, zero, zero
   jr ra
```