1. Rekursion, Strings, Dictionaries

c) Zahlen als Strings, hier nur die Additionsfunktion

```
def add(x,y):
  # Erst einfache Addition aller Stellen (Überträge ignorieren)
  z = x.copy()
  for k in y.keys():
    if k in z.keys():
      z[k] = z[k]+y[k]
    else:
      z[k] = y[k]
  # Überträge reparieren
  indizes = z.keys()          # welche Stellen hat z?
  indizes.reverse()           # bei den negativen anfangen
  for i in indizes:
    if z[i] > 9:              # Übertrag bei Pos. i?
      z[i] = z[i]-10          # dann 10 abziehen
      if i+1 in indizes:      # gibts die nächst größere
        v = z[i+1] + 1        # Position? Dann da +1;
      else:                   # anderenfalls
        v = 1                 # 1 (neuer Eintrag im Dict)
      z[i+1] = v
    if z[i] == 0:             # zuletzt noch aufräumen:
      del z[i]                # Nullen rauswerfen
  return z
```

Addieren von zwei Zahlen-Strings geht dann über mehrfache Wandlung und Aufruf von `add()`:

```
def addS (xS,yS):
  return DictToStr ( add ( StrToDict (xS) , StrToDict (yS) ) )
```

2. Pipelines

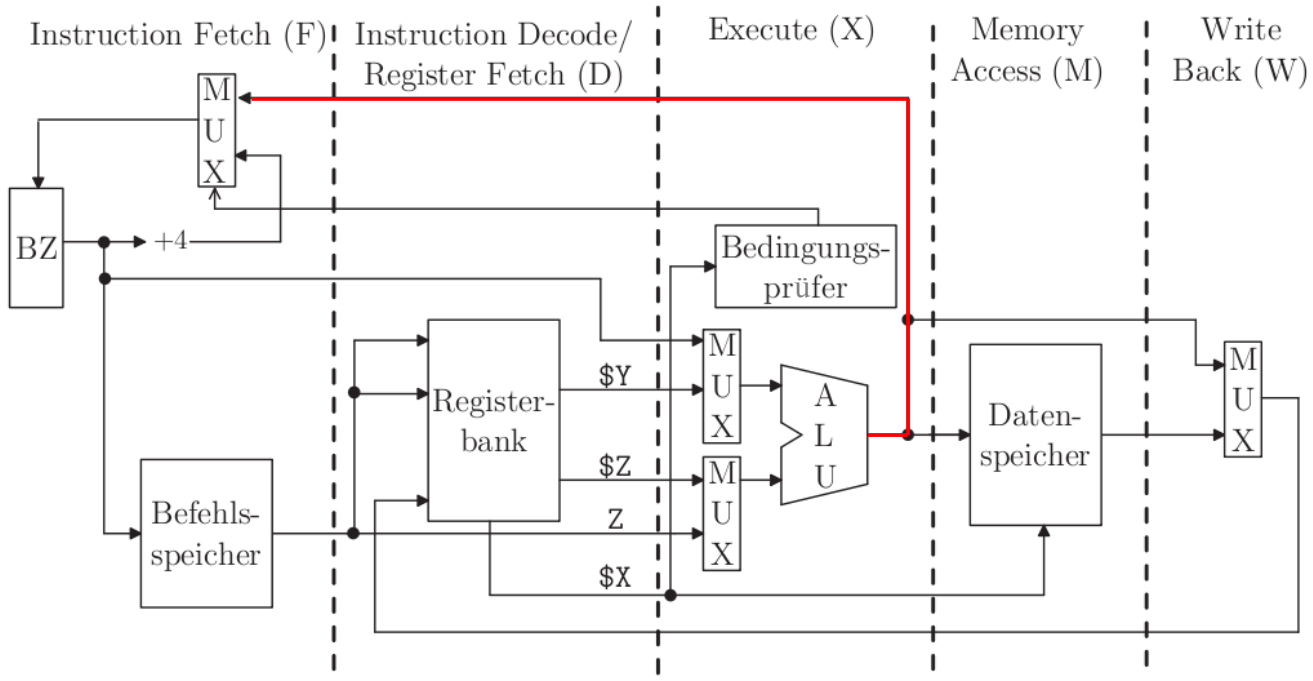
a) 6-stufige Pipeline:

1. **FI (Fetch Instruction)**
2. DI (Decode Instruction)
3. CO (Calculate Operands)
4. **FO (Fetch Operands)**
5. EI (Execute Instruction)
6. **WO (Write Operands)**

Fett markiert: Phasen mit Speicherzugriff. Kollidieren können also FI/FO, FI/WO und FO/WO:

```
FI DI CO [FO] EI WO          | FI DI CO FO EI [WO]          | FI DI CO FO EI [WO]
   FI DI [CO] FO EI WO       |    FI DI CO FO [EI] WO       |    FI DI CO FO [EI] WO
      FI [DI] CO FO EI WO    |       FI DI CO [FO] EI WO    |       FI DI CO [FO] EI WO
         [FI] DI CO FO EI WO |          FI DI [CO] FO EI WO |
                             |             FI [DI] CO FO EI WO
                             |                [FI] DI CO FO EI WO
```

b) Die JMP-Anweisung von MMIX benötigt keine W-Phase, weil das PC-Register schon nach der X-Phase geschrieben wird:



c) FXM: 2 *t*, DW: *t*; Berechnung ohne Pipelining:

Speicherzugriff mit W-Phase (Load)	FDXMW	0,25	(2+1+2+2+1) *t*	= 0,25 * 8 *t* = 2,0 *t*
Speicherzugriff ohne W-Phase (Store)	FDXM	0,10	(2+1+2+2) *t*	= 0,10 * 7 *t* = 0,7 *t*
arithmetischer Befehl	FDXW	0,55	(2+1+2+1) *t*	= 0,55 * 6 *t* = 3,3 *t*
Sprungbefehl	FDX	0,10	(2+1+2) *t*	= 0,10 * 5 *t* = 0,5 *t*

				6,5 *t*

Mit Pipelining: Abstand 2 *t*;

Beschleunigung durch Pipelining: $\frac{6{,}5t}{2\,t} = 3{,}25$