Out of Order Execution: Tomasulo

Überblick

1. Einführung in Out of Order Execution
2. Grundlagen Tomasulo Algorithmus
3. Korrektheitsbeweis
4. Erweiterung um Speichersystem

2. Grundlagen Algorithmus von Tomasulo

Kurz: Hardware Algorithmus zur automatischen Ausnutzung mehrerer Execution Units

- Gute Skalierbarkeit:
  Beliebige Anzahl Register und Execution Units

- Grundlegende Idee: Common Data Bus (CDB)

- Transport von Registerinhalten über den CDB

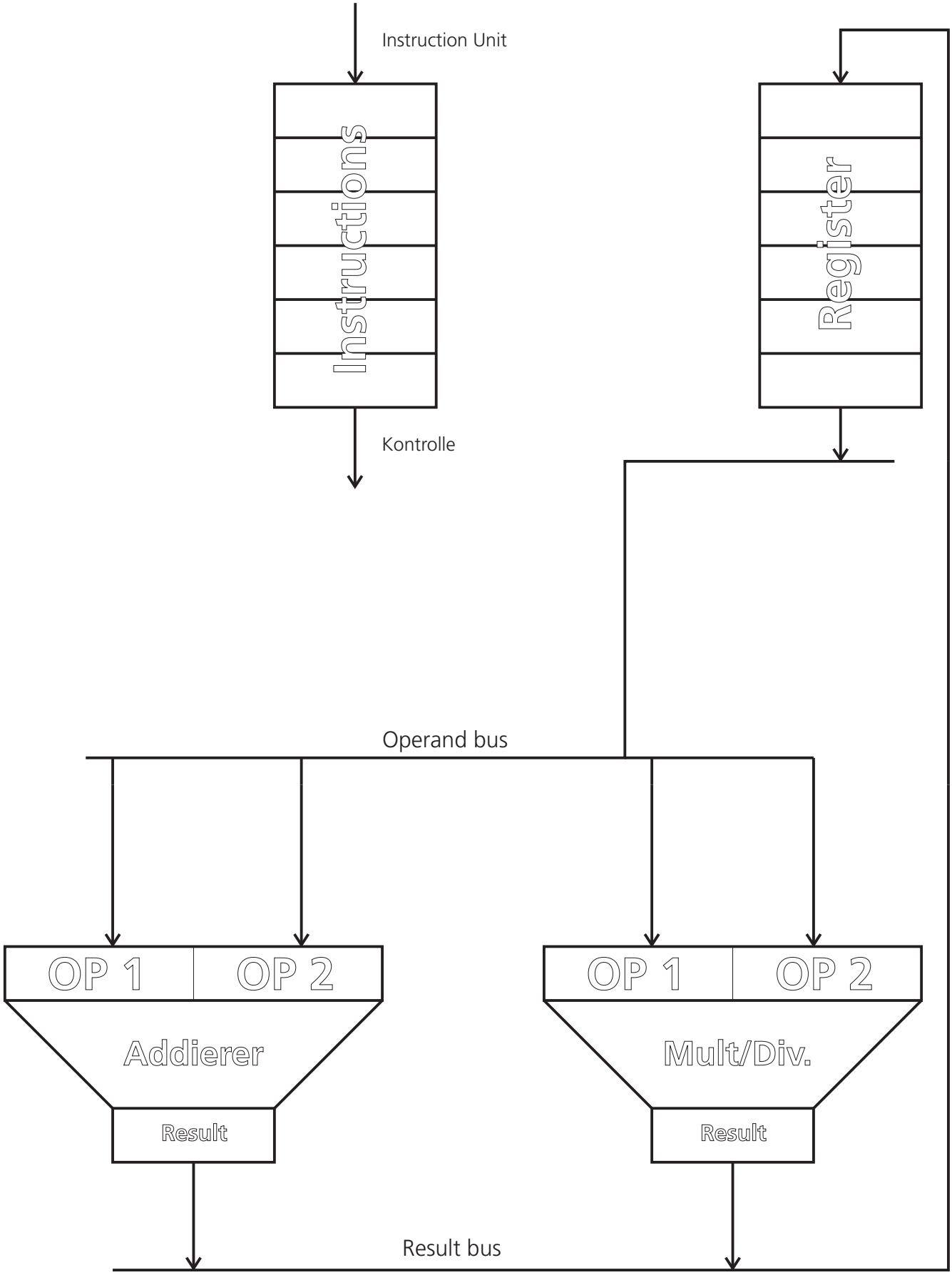
- “Reservation Stations” fordern den Bus bei einer zentralen, prioritätengesteuerten Kontrolle an.

- Identifikation der Daten nach Herkunft durch “tags”

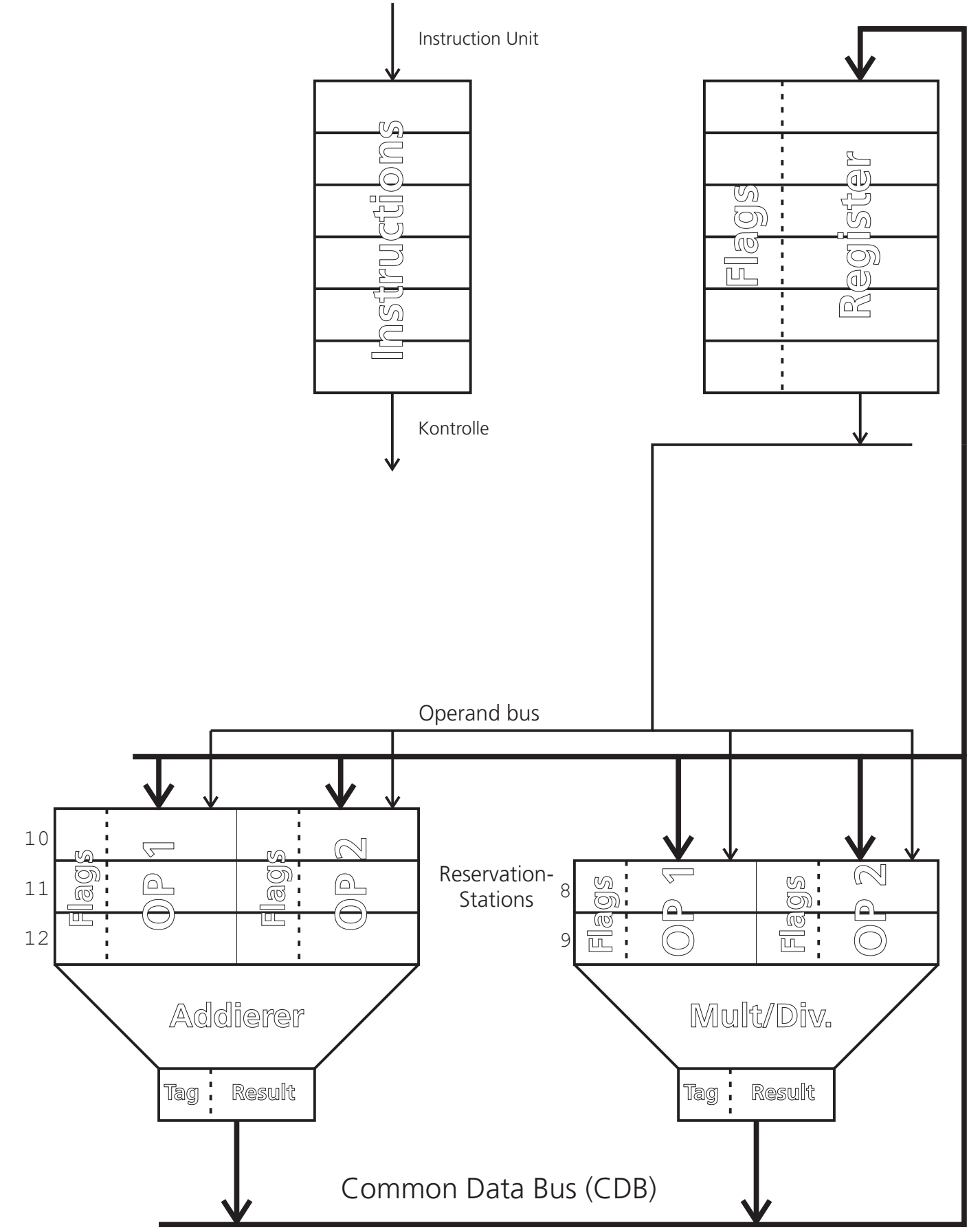
- Forwarding von Zwischenergebnissen

- keine RF-Schreibzugriffe mit veralteten Daten

Out of Order Execution: Tomasulo

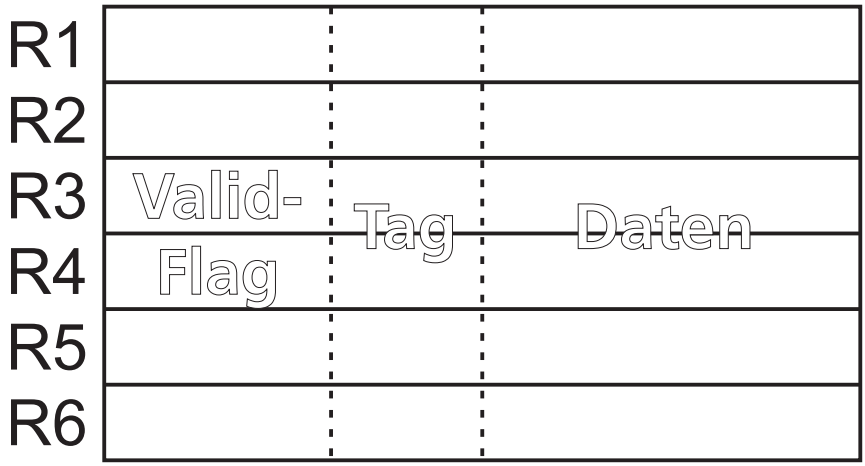


Out of Order Execution: Tomasulo



2. Grundlagen Algorithmus von Tomasulo

Erweiterungen am Register File:



Valid-Flag: 1 genau dann, wenn gültige Daten im Register

Erweiterungen an den Function Units (Reservation Station):

Tag	Dest	Op	S1.V	S1.Data	S1.Tag
Free	Busy		S2.V	S2.Data	S2.Tag

2. Grundlagen Algorithmus von Tomasulo: Protokoll

$P = \{ I_1, I_2, ..., I_p \}$ $I_i : D(i)=S1(i)\ op(i)\ S2(i)$

Ausführung durch F(i) bzw. RS(i)

Reservation Station RS:
- op (*/+- etc.)
- free/full
- Dest (Zielregister)
- busy (rechnet gerade)
- Vx, Datx, Tagx mit $x \in \{1, 2\}$

Register R im Register File:
- Tag
- V
- Data

2. Grundlagen Algorithmus von Tomasulo: Protokoll

Instruction Issue

while (I_i not issued) $\wedge$ (I_{i-1} issued)
{ if ($\exists$ r mit RS_r mit RS.op = op(i) und RS.full = 0)
{ RS(i):=RS_r ; RS(i).Adr:= D(i);
RS(i).full:=1; RS(i).busy:=0;
RS(i).op:=op(i); RS(i).Vx=Sx(i).V;
RS(i).tag:=Sx(i).tag;

D(i).tag:=r ; D(i).V:=0;

if (Sx(i).V=1) RS(i).Datx:=Sx(i).Dat (Operand Bus!)
}
}

Instruction Dispatch (Execution)

while (RS.full $\wedge$ /RS.busy)
{ if (RS.V1 $\wedge$ RS.V2)
{ FU F(i) wird angefordert
wenn F(i) frei:
{ RS.busy:=1;
RS.Datx, RS.op, RS.tag an F(i) übergeben
}
}

2. Grundlagen Algorithmus von Tomasulo: Protokoll

Write Back (unit F)

```
while ( F hat Ergebnis für RS_r )
  {  if ( CDB für F freigegeben )
       {  CDB.data:=result;   CDB.tag:= r;
          RS_r.busy:=0;  RS_r.full:=0;
          R sei das Register RS_r.Adr
          if ( R.tag=r )
           {
            R.tag:=0; R.V:=1;
            R.Dat:=CDB.data;   (d.h. result)
           }
       }
  }
```

Sender

Empfänger
(Register)

Snooping on CDB

```
while ( RS.full ∧ /RS.Vx   ∧  (CDB.tag ≠ 0))
  { if (RS.tagx = CDB.tag)
      { RS.Datx:=CDB.Dat;
        RS.Vx:=1;
        RS.tagx:=0;
      }
  }
```

Empfänger
(Operanden)

Out of Order Execution: Tomasulo

3. Korrektheit

Bezeichnungen

Wert von	D(i)	Sx(i)	(x=1,2)
sequentiell	d(i)	sx(i)	
Tomasulo	d'(i)	s'x(i)	I_i : D(i)=S1(i) op(i) S2(i)

Common Data Bus cdb(i):

Zyklus, zu dem das Ergebnis d(i) der Instruktion i auf dem CDB erscheint.

Analog dazu (Instruktion i):

decode(i)	Zyklus der Phase Decode
issue(i)	Zyklus der Phase Issue
dispatch(i)	Zyklus der Phase Dispatch

3. Korrektheit

Invarianten zur Zeit t

Register R:

$R.V = 0 \Leftrightarrow \exists\, r \neq 0$ mit $R.tag = r$
$\wedge\ \exists\, i : issue(i) < t \leq cdb(i) \ \wedge D(i) = R \wedge RS(i) = RS_r$
$\wedge\ \forall\, k>i : (D(i) \neq R \vee issue(k) \geq t)$

$R.V=1 \Leftrightarrow R.tag = 0 \wedge \exists\, j : cdb(j) < t \wedge D(j) = R$
$\wedge\ R.dat = d'(j)$ (= cdb(j).data)
$\wedge\ \forall\, k>j : (D(i) \neq R \vee issue(k) \geq t)$

Reservation Station RS:

$RS.full = 1 \Leftrightarrow \exists\, i : RS(i) = RS \wedge issue(i) < t \leq cdb(i) \wedge RS.Dest = D(i)$

$RS.busy = 1 \Leftrightarrow \exists\, i : RS(i) = RS \wedge dispatch(i) < t \leq cdb(i) \wedge RS.Dest = D(i)$

Sei RS die Reservationstation zu Is und RS.full = 1:

$RS.Vx = 0 \Leftrightarrow R.tagx = r \neq 0$
$\wedge\ \exists\, i < s : issue(i) < t \leq cdb(i) \wedge D(i) = Sx(s) \wedge RS(i) = RS$
$\wedge\ \forall\, i<k<s : D(i) \neq D(k)$

$RS.Vx = 1 \Leftrightarrow R.tagx = 0$
$\wedge\ \exists\, i < s : cdb(i) < t \wedge D(i) = Sx(s) \wedge RS.datx = d'(i)$
$\wedge\ \forall\, i<k<s : D(i) \neq D(k)$

3. Korrektheit

z.z.: a) Datenkonsistenz
b) Terminierung (no deadlocks)

Datenkonsistenz

z.z.: $\forall$ i gilt: $d(i)=d'(i)$ und $sx(i)=s'x(i)$

Bew. durch Ind. über die Zeit t (Zyklen):

IA: $t=0$ Ok (Reset)

IS: $t \rightarrow t+1$

Writes: Def. j mit $cdb(j) = t+1$
$\Rightarrow$ Reads (d.h. Source-Operands) von Befehl I_j zur Zeit t oder früher
$\Rightarrow$ (IVor, Korrektheit der FUs) $d'(i) = cdb(j).dat = d(i)$

Reads: I_i liest Operand $Sx(i)$ und $R:=Sx(i)$
a) Operand über Operand Bus:
$t+1 = issue\ (i) \wedge R.V=1$
I_k sei die letzte Instr. (d.h. $k<i$), die R geändert hat.
$R.dat=d(k)=d'(k) \wedge \forall h: k<h<i$ gilt: $D(h)\neq R$

b) Operand über CDB:
$t+1 > issue\ (i) \wedge RS(i).tagx=cdb(j).tag$ (Zeit: $cdb(j)=t+1$)
$cbd(j).dat = d(j) = d'(j) \wedge D(j)=Sx(i) \wedge \forall h: j<h<i: D(h)\neq D(j)$

$\Rightarrow s'x(i)=sx(i)$

Out of Order Execution: Tomasulo

3. Korrektheit

Terminierung

1. Sei T(i) die kleinste Zahl t mit:
   alle Instruktionen $I_1 ... I_{i-1}$ sind bis zum Zyklus t terminiert

① $\Rightarrow T(i+1) = \max\{cdb(i), T(i)\}$

z.z.: $\exists\, \alpha$ mit $T(i+1) \leq T(i) + \alpha \leq i\, \alpha < \infty$

2. CDB-Schedule: Round-Robin

Sei f = # units. CDB-Request zur Zeit t

② Acknowledge spätestens zu t+f-1

3. Beweis

③ $issue(i) \leq T(i) + 1$

④ $dispatch(i) \leq \max\{ issue(i), T(i) \} + 1 + Emax + f$

$= T(i) + 2 + Emax + f$ (wg. ③)

$cdb(i) \leq dispatch(i) + \underbrace{(Emax-1) \cdot f}_{\text{Ann.: Pipeline voll}} + 1 + \underbrace{f-1}_{\text{CDB}}$ (wg. ②)

$\leq T(i) + \underbrace{2 + Emax\,(f+1) + f}_{\alpha}$ (wg. ④)

Emax: Maximale Dauer einer Berechnung einer FU

Out of Order Execution: Tomasulo

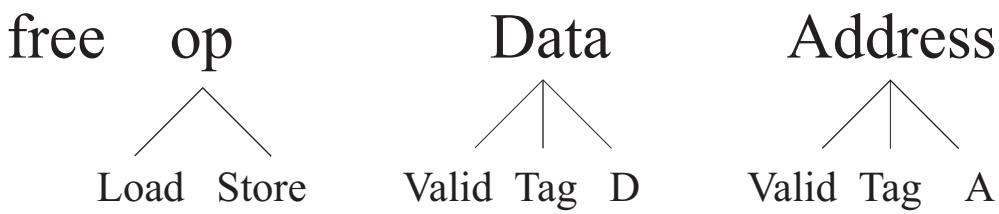
4. Erweiterung um Speichersystem

Load-Buffer

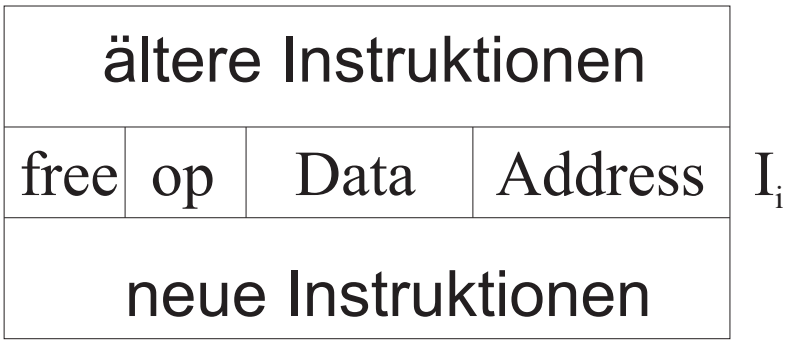
- Jeder Eintrag im Load Buffer erhält ein Tag

- Flags: free, valid, data, Dest

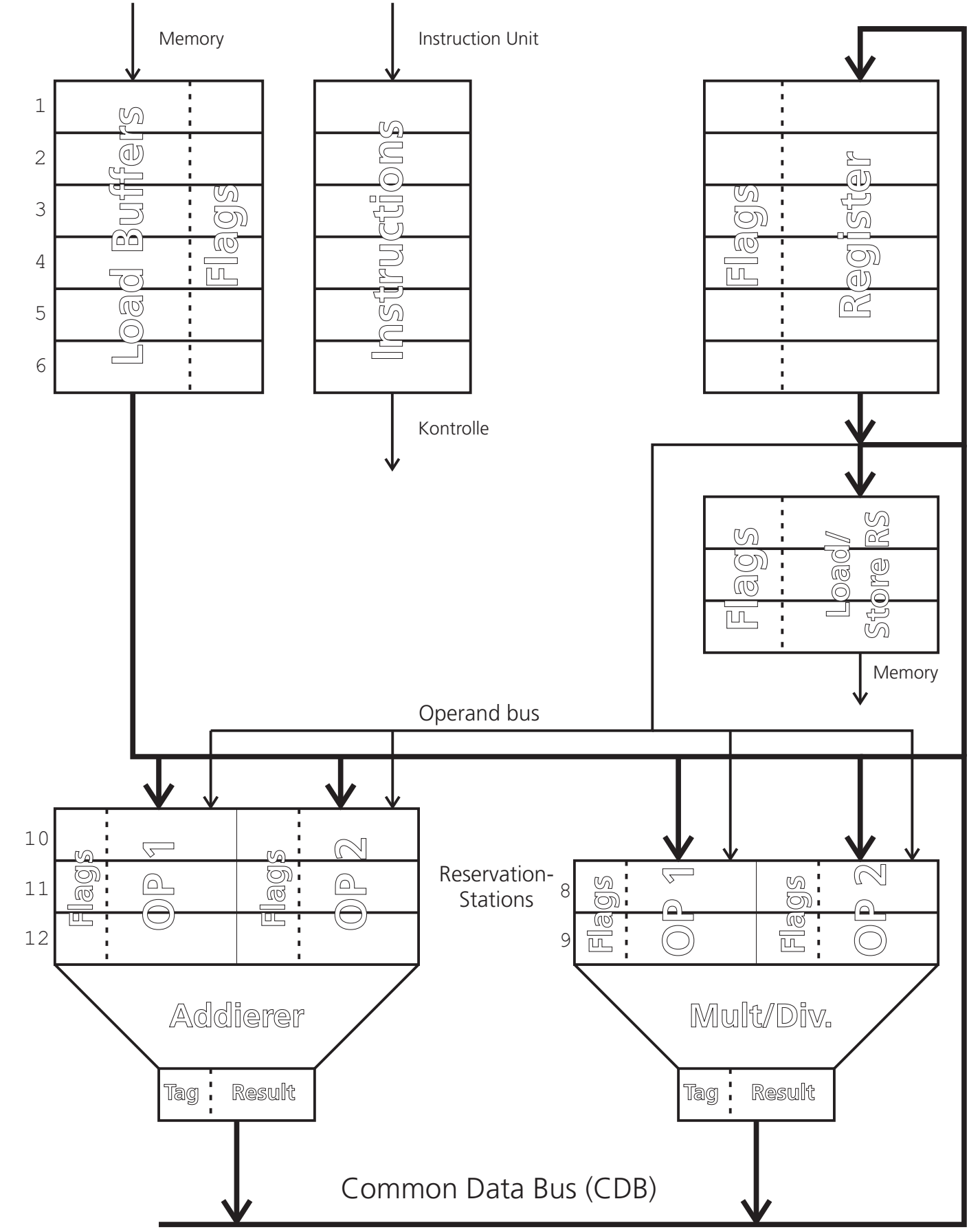
Load/Store-Reservation Station



Für Loads: Data.Valid:=1, Data.Tag:=Tag des Load Buffer



Out of Order Execution: Tomasulo



4. Erweiterung um Speichersystem

Store-Dispatch

(1) $Data.Valid = Address.Valid = 1$

(2) $\forall j<i : I_j.Address.Valid = 1$

(3) $\forall j<i : I_j.Address.Data \neq I_i.Address.Data$

Load-Dispatch

(1) $Data.Valid = Address.Valid = 1$

(2) $\forall j<i : I_j.Address.Valid = 1 \vee I_j.op = Load$

(3) $\forall j<i : I_j.Addr.Data \neq I_i.Addr.Data \vee I_j.op = Load$

Optimierung: Forwarding

(4) (1-3) $\wedge \exists j<i$: ($I_j.Address.A = I_i.Address.A \wedge I_j.Data.Valid = 1 \wedge$
$I_j.Address.Valid = 1 \wedge \forall k: j<k<i : I_j.Address.A \neq I_k.Address.A$)

$\Rightarrow I_j.Data.D$ an Load Buffer weiterreichen