

Instruction Set Architecture **(ISA)**

Peter Marwedel
Informatik 12
TU Dortmund

2011/04/07

Diese Folien enthalten Graphiken mit
Nutzungseinschränkungen. Das Kopieren der
Graphiken ist im Allgemeinen nicht erlaubt.

2.2 DSP-Befehlssätze

DSP = *Digital Signal Processing*

Spezialanwendung für Rechner, in der Regel in **eingebetteten** Systemen (*embedded systems*), IT ist in eine Umgebung eingebettet z.B.:

- im Telekommunikationsbereich (Mobiltelefon),
- im Automobilbereich (Spurhalteassistent),
- im Consumerbereich (Audio/Video-Komprimierung)



Wichtige Teilaufgabe: (Digitale) Signalverarbeitung

Dabei: Effizienz und Realzeitverhalten extrem wichtig!



☞ Spezielle Strukturen / Befehle

DSP: Digitale Filterung

Signalverarbeitungsmodell



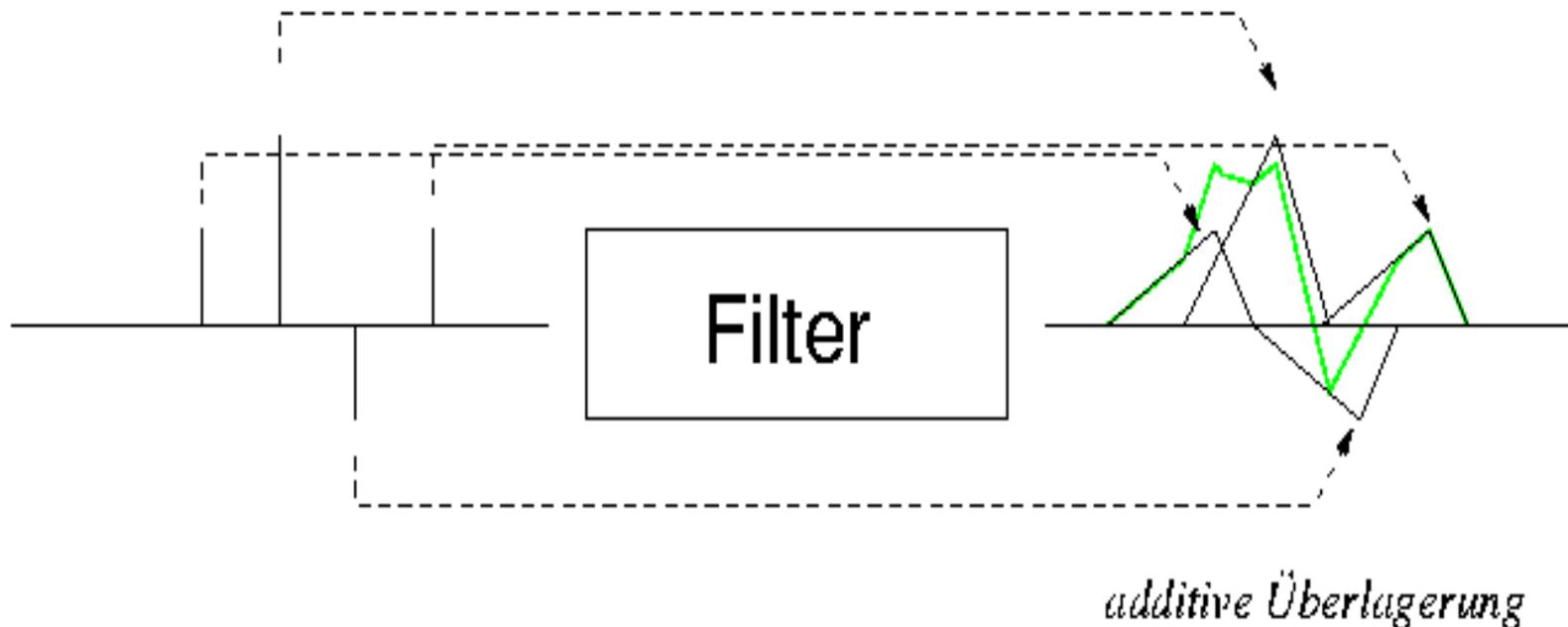
w und x sind Signale
(mathematisch: Abbildungen von der Zeit auf Signalwerte)

Unter bestimmten Einschränkungen (lineares System) lässt sich das Verhalten des Filters durch die sog. Impulsantwort beschreiben, d.h. die „Antwort“ auf einen Einheitsimpuls



DSP: Digitale Filterung 2

Für lineare Systeme ist Transformation, die ein Filter realisiert, durch Faltung berechenbar
(d.h. Faltung des Eingangssignals mit der Impulsantwort)



DSP: Digitale Filterung 3

Mathematische Formulierung:

Impulsantwort: Diskrete, endliche Folge (hier: kausal!)

$$a(k), \quad k=0 \dots n-1$$

Filterung des Signals w , mit $w(s) =$ Wert von w zur (*sampling*-) Zeit t_s :

$$x(s) = \sum_{k=0}^{n-1} a(k) \cdot w(s-k)$$

Ergebnis kann iterativ aus Partialsummen berechnet werden

$$x_k(s) = x_{k-1}(s) + a(k) \cdot w(s-k)$$

Randbedingungen

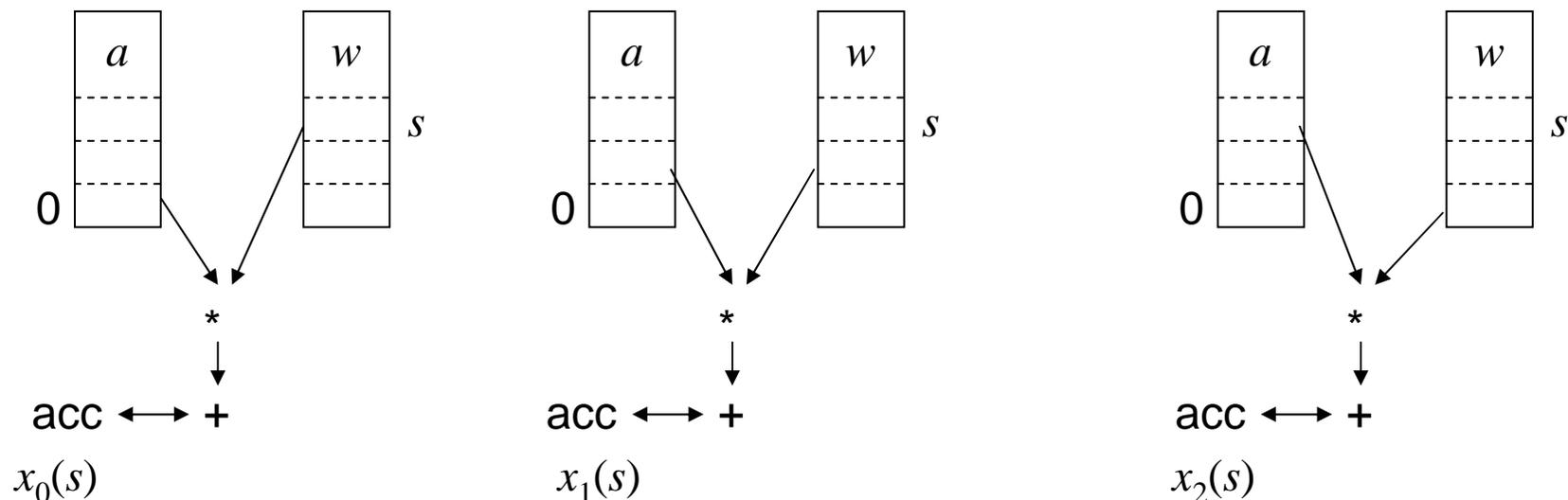
$$x_{-1}(s) = 0 \text{ und } x(s) = x_{n-1}(s)$$

☞ Akkumulation von Produkten!

Multiply-Accumulate-Befehl

Effizienz bei DSP von zentraler Bedeutung!

☞ Spezieller Befehl (*Multiply Accumulate* [MAC]) für genau diese Berechnungsstruktur; Vorgehen:

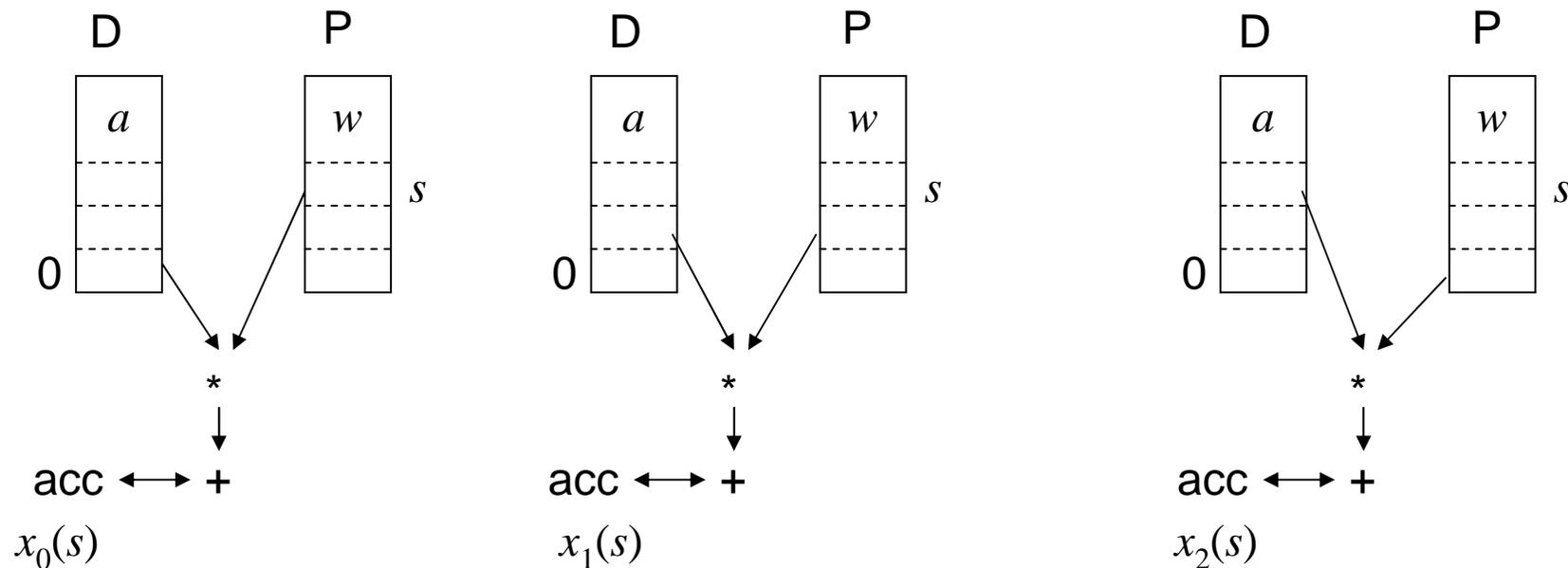


Funktionalität des *Multiply-Accumulate*-Befehls

MAC-Befehl muss in einem Zyklus leisten:

1 Multiplikation, 1 Addition, 2 Speicherzugriffe, ...

👉 Abbildung auf 2 Hardwarespeicher (D und P)



Zusätzlich: Aktualisierung der Indexregister notwendig!

Multiply/accumulate (MAC) und zero-overhead loop (ZOL) Befehle

```
MR:=0; A1:=1; A2:=s-1; MX:=w[s]; MY:=a[0];
```

```
for ( k:=0 <= n-1)
```

```
{MR:=MR+MX*MY; MY:=a[A1]; MX:=w[A2]; A1++; A2--}
```

Multiply/accumulate (MAC) Befehl

*Zero-overhead loop (ZOL) Befehl
vor dem MAC Befehl.
Schleifentest erfolgt parallel zu
den MAC-Operationen.*

Multiply-Accumulate-Befehl

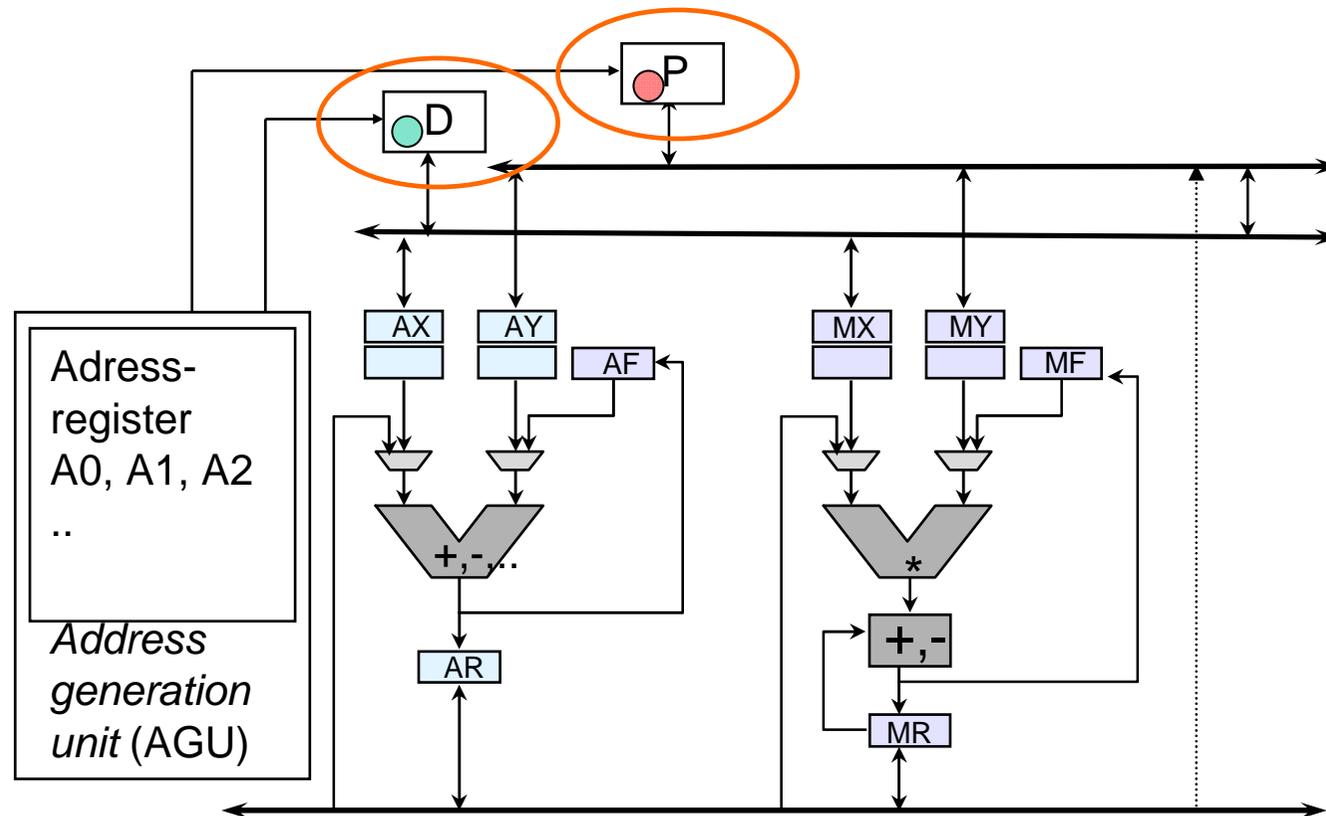
DSP-Code-Beispiel (Intel 80296SA):

```
LDB ICB0,#01H      ;set up increment control byte reg
LDB ICB1,#01H
LD  IDX0,SAMPLE    ;init sample pointer
LD  IDX1,COEFF     ;init coefficient pointer
SMACZ ICX0,ICX1    ;do initial MPY, zero acc
RPT #0EH          ;repeat next instr 15x
SMACR ICX0,ICX1   ;do 15 successive MACs with inc
MSAC YOUT,#018H   ;place results in YOUT
```

nach N. Govind, Intel

Inkremente der Indexregister konfigurierbar (ICB0 / 1)!
Schleife mit fester Interationszahl via **RPT**

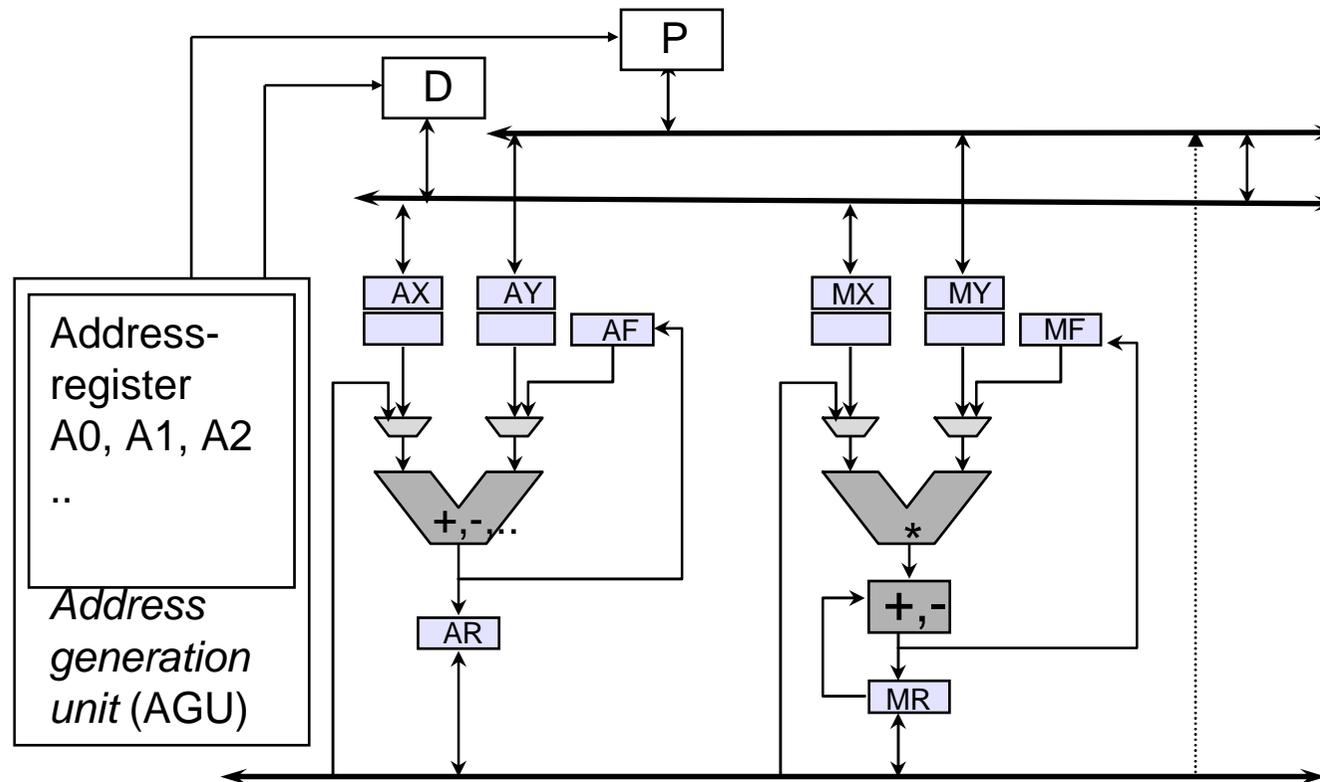
Mehrere Speicherbänke oder Speicher



Vereinfacht paralleles Holen von Daten

Heterogene Register

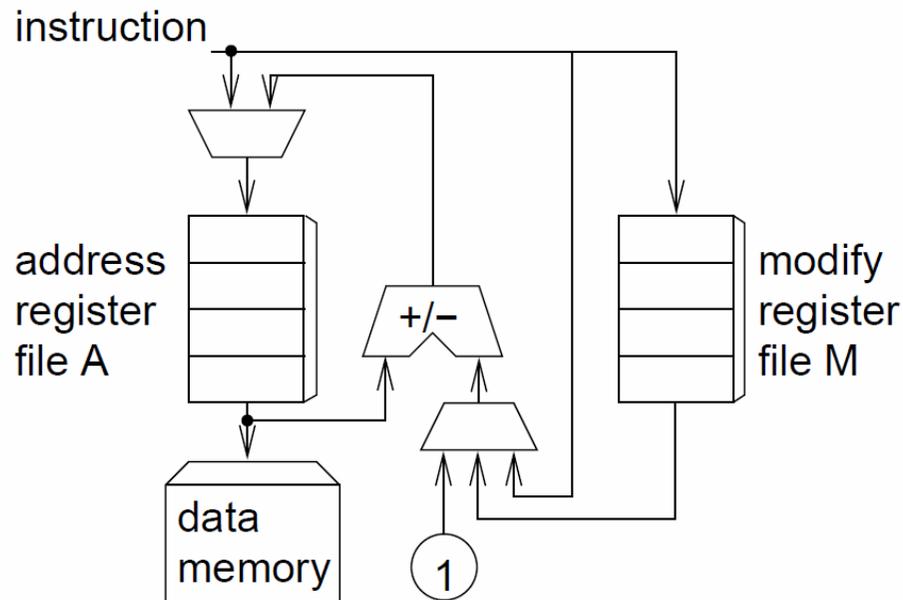
Beispiel (ADSP 210x):



Verschiedene Funktionalität der Register AX, AY, AF, MX, MY, MF und MR

Separate Adresserzeugungseinheiten (AGUs)

Beispiel (ADSP 210x):



- Datenspeicher kann nur mit der in A enthaltenen Adresse gelesen werden,
 - Dies ist parallel zur Operation in der Haupt-ALU möglich **(kostet effektiv keine Zeit)**.
 - $A := A \pm 1$ ebenfalls in Zeit 0,
 - dsgl. für $A := A \pm M$;
 - $A := \langle \text{immediate in instruction} \rangle$ bedarf eines extra Befehls
- ☞ Wenige *load immediates*!

Speicherung von Signalen

Problem: Signale = zeitlich *fortschreitende* Folgen von (digitalen) Messwerten, d.h. potentiell *unendlich*!

Lösung: Speicherung und Bearbeitung nur eines (relativ) kurzen Ausschnitts („Fenster“)

Zu jedem Zeitpunkt ...

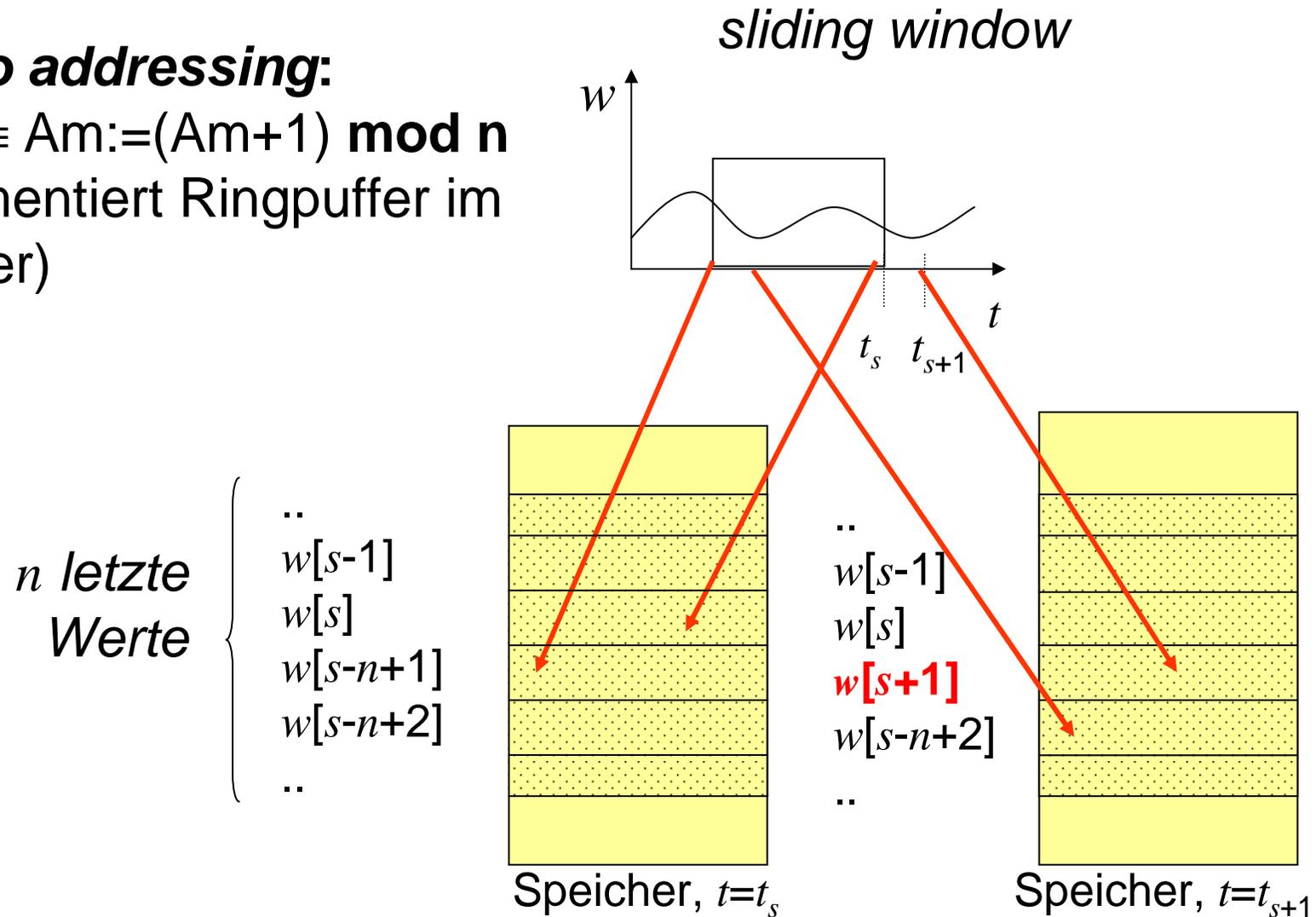
- ... trifft neuer Messwert ein  speichern
- ... und fallen „alte“ Werte aus dem Betrachtungsbereich heraus  verwerfen

Speicherstellen können wieder verwendet werden!
Wie möglichst effizient Daten verwalten?

Modulo addressing

Modulo addressing:

$A_{m++} \equiv A_m := (A_{m+1}) \bmod n$
 (implementiert Ringpuffer im Speicher)



Modulo-Adressierung

Programmbeispiel zur Filterung:

```
s:=0;           % initialer Zeitpunkt
repeat
  w[s] %in D[&w[s]]% := nächster Eingabewert
  acc := 0;
  A1:=&a[0]; A2:=&w[s];
  x:=D[A1]; Y:=P[A2]; A1++; A2--;
  for k:=0 to n-1 do
    {acc := acc + X*Y,
     X:=D[A1], Y:=P[A2],
     A1++, A2--}      % acc=x[s]
  s:=s+1;           % wächst unbeschränkt
until false;       % d.h. Endlosschleife
```

Modulo-Adressierung 2

Programmbeispiel zur Filterung:

```
s:=0;                % initialer Zeitpunkt
repeat
  w[s] %in D[&w[s]]% := nächster Eingabewert
  acc := 0;
  A1:=&a[0]; A2:=&w[s];
  X:=D[A1]; Y:=P[A2]; A1++; A2:=(A2-1)mod n;
  for k:=0 to n-1 do
    {acc := acc + X*Y,
     X:=D[A1], Y:=P[A2],
     A1++, A2:=(A2-1)mod n;}           % acc=x[j]
  s:=(s+1) mod n;           % läuft zyklisch durch Puffer
until false;              % d.h. Endlosschleife
```

Fourier-Transformation

- Fourier-Transformation = Darstellung eines Signals im Frequenzbereich (im Prinzip: durch Überlagerung skaliertes und phasenverschobener Sinusschwingungen)
- Diskrete Fourier-Transformation: Für diskrete, periodische Signale
☞ kann auf Daten einer Periode berechnet werden;
hier im 1-dimensionalen Fall:

$$X(\nu) = \frac{1}{N} \sum_{s=0}^{N-1} x(s) \cdot e^{-i2\pi \frac{\nu \cdot s}{N}}$$

- FFT (*Fast Fourier Transformation*):
Schnelle Berechnung für $N=2 \cdot M$ durch rekursive Zerlegung in geraden und ungeraden Anteil
☞ Effizienz $O(N \log N)$

Fast Fourier-Transformation (FFT)

FFT ergibt folgendes Berechnungsschema:

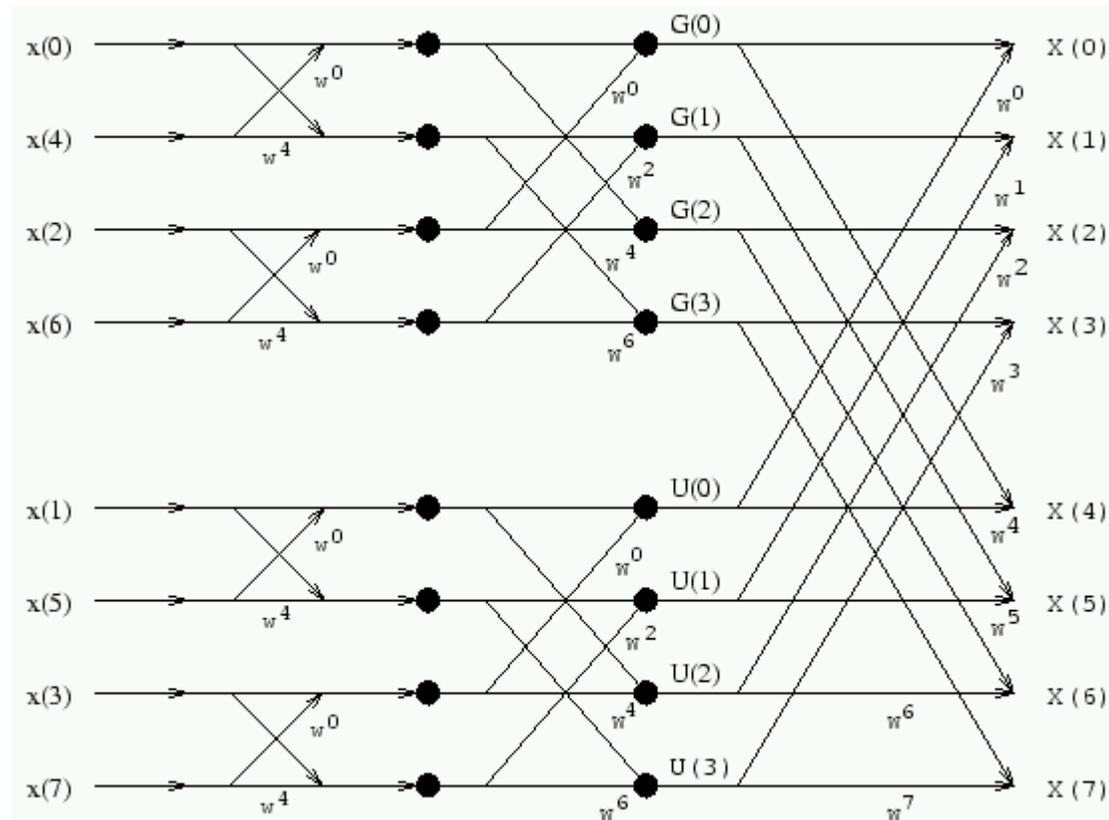
hier für FFT mit $N=8$

Aufteilung in
gerade und un-
gerade Indices

“Umsortierung”
der Eingabedaten:

$$x(4) - x(1)$$

$$x(3) - x(6)$$



Bit-Reversal

„Umsortierung“ der Eingangsfolge bei der FFT erfolgt gemäß dem sog. *Bit-Reversal*

- Bei m bit Adressbreite $a_m a_{m-1} \dots a_2 a_1 a_0$ ergibt sich neue Adresse durch Umkehrung der Bitreihenfolge: $a_0 a_1 a_2 \dots a_{m-1} a_m$
- Beispiel (4 bit Adressen):
 $(0000)_2 \rightarrow (0000)_2$
 $(0001)_2 \rightarrow (1000)_2$
 $(0010)_2 \rightarrow (0100)_2$
 $(0011)_2 \rightarrow (1100)_2$

☞ spezielle Adressierungsart mit *Bit-Reversal*
(spez. Betriebsart eines der Adressgeneratoren [ADSP219x])

DSP: *Bit-reverse* Adressierung

- DSP-Code-Beispiel (ADSP 219x) -

```
br_adds: I4=read_in;           % DAG2 ptr to input
        I0=0x0200;           % Base addr of bit_rev output
        M4=1;                % DAG2 increment by 1
        M0=0x0100;           % DAG1 incr. for 8-bit rev.
        L4=0;                % Linear data buffer
        L0=0;                % Linear data buffer
        CNTR=8;              % 8 samples
        ENA BIT_REV;         % Enable DAG1 bit rev. mode
        DO brev UNTIL CE;
            AY1=DM(I4+=M4); % sequential read
        brev: DM(I0+=M0)=AY1; % bit reversed write
        DIS BIT_REV;         % Disable DAG1 bit rev. mode
        RTS;                 % Return to calling routine
read_in: % input buf, could be .extern
        NOP;
```

nach ADSP-219x/2191 DSP Hardware Reference

Problem der *wrap around* Arithmetik

- Problem: Ergebnisse von Berechnungen mit Bereichsüberschreitungen mit *wrap around* sind ..
... nicht nur **falsch**
... sondern extrem **unplausibel** /
nicht einmal nahe der korrekten Lösung
- Der notwendigerweise entstehende Fehler ist **maximal**
(signifikanteste Stelle 2^n geht verloren) nicht minimal! z.B.:

$$(4 \text{ bit, 2er Kompl.}): \quad |(7 +_{\text{wrap}} 1) - (7 +_{\text{exact}} 1)| = |-8 - 8| = 16$$

- Große Fehler zwischen (mit Überlauf) berechnetem und tatsächlichem Ergebnis besonders dramatisch bei Signalverarbeitung (Verstärkung eines Audiosignals / Helligkeitsänderung eines Bildpunktes)

Kleinerer Fehler bei Sättigungsarithmetik

Sättigungsarithmetik (für Addition / Multiplikation) liefert bei **Über-/Unterlauf jeweils maximal/minimal darstellbaren Zahlenwert:**

Beispiele:

- Betragsdarstellung (4 bit):

$$8 + 8 \rightarrow 15 \neq 16$$

$$7 + 11 \rightarrow 15 \neq 18$$

- 2er-Komplementdarstellung (4 bit)

$$7 + 1 \rightarrow 7 \neq 8$$

$$-5 - 7 \rightarrow -8 \neq -12$$

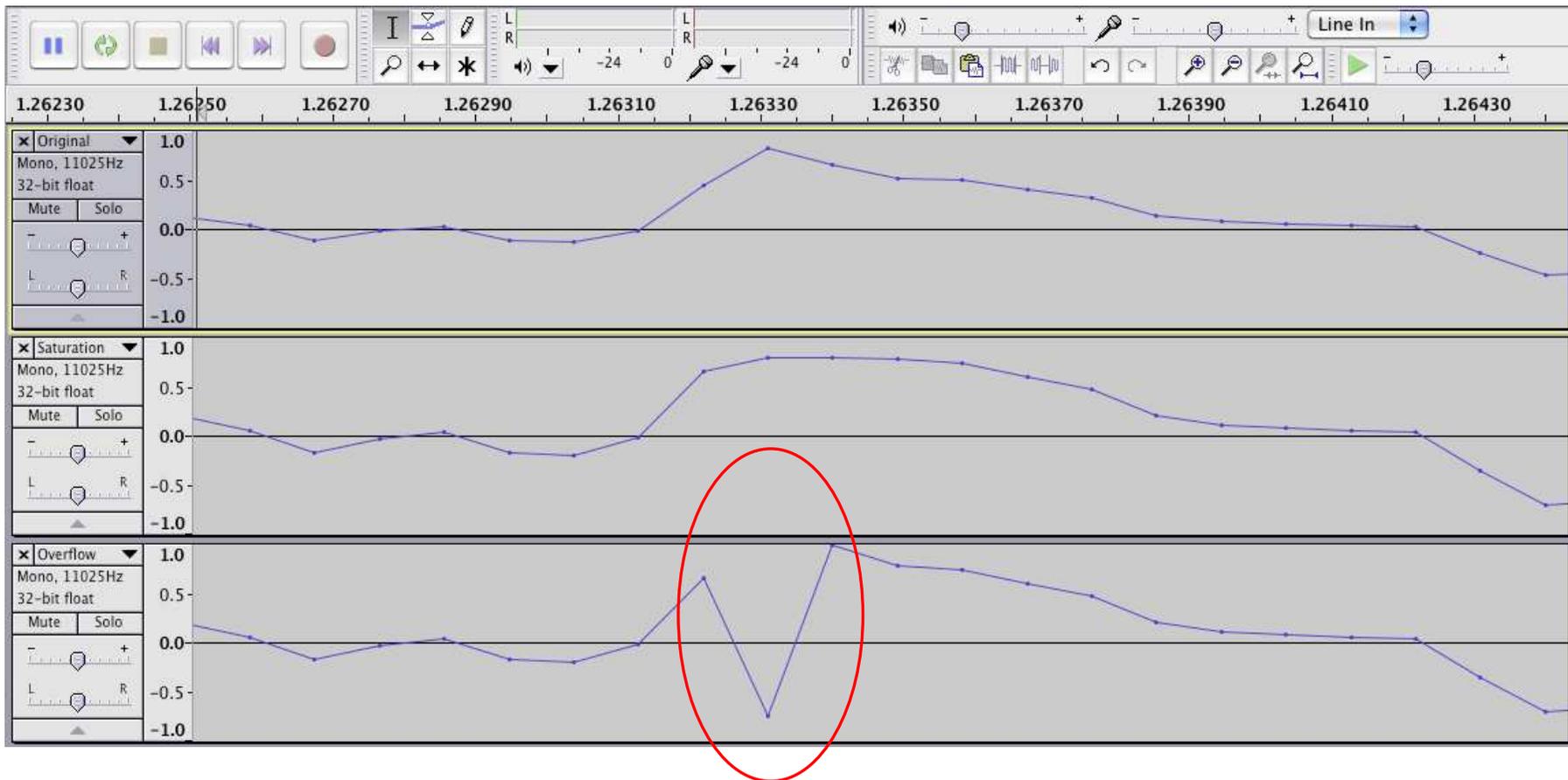
Insbesondere keine Vorzeichenumkehr!

Weiteres Beispiel

■ a		0111
b	+	1001
Standard <i>wrap around</i> Arithmetik		(1)0000
Sättigungsarithmetik		1111
(a+b)/2: korrekt		1000
<i>wrap around</i> Arithmetik		0000
Sättigungsarithmetik + >>		0111 “fast richtig“

- Geeignet für DSP/Multimedia-Anwendungen:
 - Durch Überläufe ausgelöste Interrupts  Zeitbedingungen verletzt?
 - Genaue Werte ohnehin weniger wichtig
 - *wrap around* Arithmetik liefert schlechtere Ergebnisse.

Audio-Beispiel



Sättigungsarithmetik: Bewertung

Vorteil:

- Plausible Ergebnisse bei Bereichsüberschreitungen

Nachteile:

- Aufwendiger in der Berechnung
- Assoziativität etc. sind verletzt

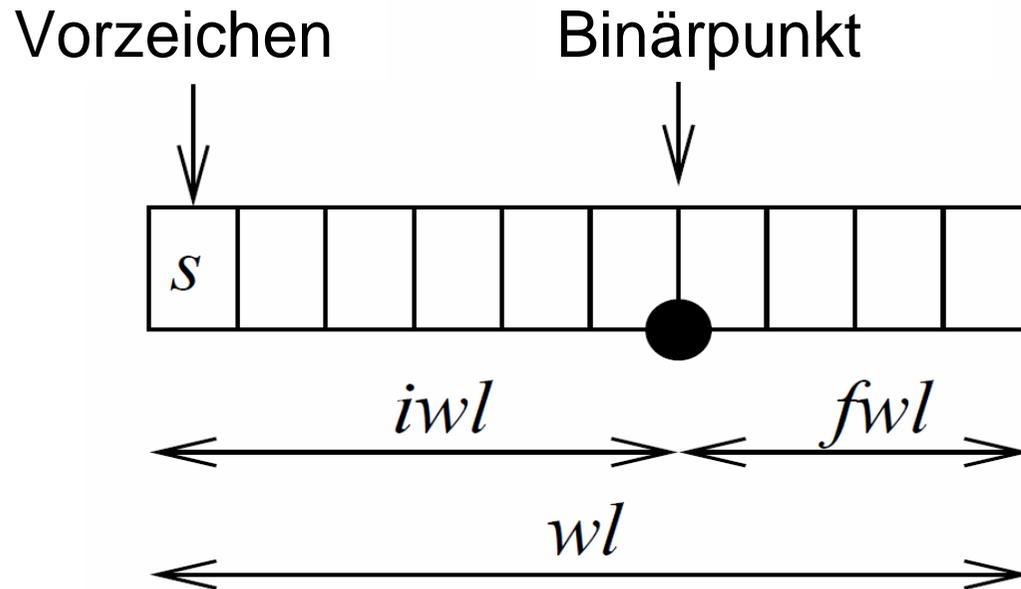
Sättigungsarithmetik und „Standardarithmetik“ können auf DSPs in der Regel wahlweise benutzt werden (ex. entsprechende Befehlsvarianten)

„Sättigung“ im IEEE 754 FP-Standard

- Bei Über-/Unterlauf entsteht +/- “unendlich” als Ergebnis
- Weitere Operationen ändern diesen Wert nicht mehr!

☞ nach Überschreitung entstehen nicht ggf. wieder *augenscheinlich gültige* Ergebnisse

Festkommaarithmetik



Nach Multiplikationen und Divisionen sind Schiebeoperationen erforderlich, um wieder die richtige Zahl an Nachkommastellen zu haben

Eigenschaften der Festkommaarithmetik

- Automatische Skalierung vorteilhaft für Multiplikationen

- Beispiel:

$$\begin{aligned}x &= 0.5 \times 0.125 + 0.25 \times 0.125 = 0.0625 + 0.03125 \\ &= 0.09375\end{aligned}$$

Für $iwl=1$ und $fwl=3$ Dezimalstellen: weniger signifikante Stellen werden automatisch abgeschnitten: $x = 0.093$

Wie ein Gleitkommasystem mit Wertebereich $\in (-1..1)$, ohne gespeicherten Exponenten (Bits werden benutzt, um die Genauigkeit zu erhöhen).

- Für Multimediaanwendungen angemessen, da dort die Wertebereiche bekannt sind.

Realzeiteigenschaften

- **Das Zeitverhalten sollte vorhersagbar sein**
Eigenschaft, die Probleme verursachen:
 - Zugriff zu gemeinsamen Ressourcen
 - *Caches* mit Ersetzungsstrategien mit problematischem Zeitverhalten
 - *Unified caches* (Konflikte zwischen Daten und Befehlen)
 - Fließbänder (*pipelines*) mit *stall cycles* ("*bubbles*")
 - *Multi-cores* mit unvorhersagbaren Kommunikationszeiten
 - Sprungvorhersage, spekulative Ausführung
 - Interrupts, die zu jedem Zeitpunkt möglich sind
 - Speicherauffrischen (*refresh*) zu jeder Zeit
 - Befehle mit datenabhängigen Ausführungszeiten
- ☞ **So viele dieser Eigenschaften vermeiden, wie möglich**

DSP-Befehlssätze: Zusammenfassung

- Spezielle Befehle für Anwendungs-Kernels, z.B. *Multiply Accumulate* (MAC), für Filterung
- Heterogene Registersätze
Zur Unterstützung spezieller Befehle (z.B. MAC)
- Eingeschränkte Parallelität
z.B. Transfer- und Adressop. parallel zu ALU-Op
(siehe MAC)
- Spezielle Adressierungsarten, z.B.:
 - Modulo-Adressierung (z.B. für Ringpuffer)
 - *Bit-Reversal* (für Fourier-Transformation)
- Sättigungsarithmetik
- Realzeitfähigkeit
(d.h. meist kein Cache, virtueller Speicher)