1 Multiplication of 2 n-bit numbers gives a $2n$ bit produc

$(3)_{10} = (011)_{2's}$
$(2)_{10} = (010)_{2's}$
(simple since both positive)

```
  011
×010
 000
011
000
000110 = (6)10
```

$(-3)_{10} = (101)_{2's}$
$(2)_{10} = (010)_{2's}$

```
  111101
×000010
 000000
111101
000
000
00
0
111010 = (−6)10
```

$(3)_{10} = (011)_{2's}$
$(-2)_{10} = (110)_{2's}$

```
  000011
×111110
 000000
000011
000011
000011
00011
00001
111010 = (−6)10
```

use sign extension to 2n bits

Dynamic range = $20\log_{10}\left(\frac{\text{largest number}}{\text{smallest positive number excluding 0}}\right)$

Example: Q0.15 format
Largest number = $(0.111111111111111)_{2's} = 1 - 2^{-15}$
Smallest number = $(0.000000000000001)_{2's} = 2^{-15}$
Dynamic range = 90 dB

Example: Q15.0 format
Largest number = $(0111111111111111)_{2's} = 2^{15} - 1$
Smallest number = $(0000000000000001)_{2's} = 1$
Dynamic range = 90 dB

In fact, any other 16-bit 2's complement Q$m.n$ format, also has a dynamic range of 90 dB

Addition of fixed point numbers:
Adding two N-bit Q$m.n$ numbers requires (N+1)-bit Q(m +1).n format to avoid overflow

Multiplication of fixed point numbers:
Multiplying two N-bit Q$m.n$ numbers requires 2N-bit Q(2m+1).(2n) format to avoid overflow

Example: Q3.0 format

```
  7        (0111)2's
x 6        (0110)2's
 42  (00101010)2's
```

Q0.3 format

```
  0.875    (0.111)2's
x 0.75     (0.110)2's
 0.65625 (00.101010)2's
```

In IEEE 754 exp is biased
example, for E = 8, The exponent bias is 2^7-1 = 127. I
exponent is –3, it will be recorded as –3 + 127 = 124, ...
$(01111100)_2$

IEEE 754 single precision

	Exponent	Mantissa
Normalized numbers	1 to $2^E - 2$ (254), biased binary	Any number
Denormalized numbers	0	Non-zero
Zeroes	0	0
Infinities	$2^E - 1$ (255)	0
NaNs	$2^E - 1$ (255)	Non-zero

For example, given | 0 | 10000010 | 10000000

Sign S = 0
Exponent Exp = $(10000010)_2$ = 130
Denormalized mantissa D = $(100\ldots)_2$
Mantissa M = $(1.100\ldots)_2$ = 1.5
So, value = $(-1)^0 \times 2^{130-127} \times 1.5 = 12$

Quantized
Typically, a quantizer is mid-tread (origin on tread of staircase)
- Number of quantizer levels = $2^n - 1$
- Quantization step size $Q = \frac{V_{max} - V_{min}}{2^n - 1}$

A mid-rise quantizer (origin on rise of staircase)
- Number of quantizer levels = 2^n
- Quantization step size $Q = \frac{V_{max} - V_{min}}{2^n}$

- The probability density function for $e(n)$ is uniform from $-Q/2$ to $Q/2$
- mean of $e(n)$ = 0
- maximum $e(n) = Q/2$
- mean square error = variance of $e(n) = Q^2/12$
- root mean square error = std dev of $e(n) = 0.29Q$

Signal to Quantization Noise Ratio (SQNR)

- Range of input signal = $-V_{max}$ to V_{max}
- Signal variance = σ_x^2
- Quantization noise variance = σ_e^2

$$SQNR = 10\log\left(\frac{\sigma_x^2}{\sigma_e^2}\right) = 10\log\left(\frac{\sigma_x^2}{Q^2/12}\right) = 10\log\left(\frac{3\sigma_x^2 2^{2n}}{V_{max}^2}\right) = 6.02n + 4.77 + 20\log(\sigma_x/V_{max})$$

SQNR improves by 6 dB per additional bit of ADC

SQNR can also be improved by over-sampling and low-pass filtering. Doubling the sampling rate improves the SQNR by 3 dB.

Advantages and disadvantages of dithering

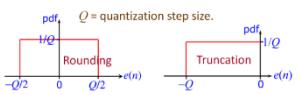
Advantge:
- Adding noise provides more information!
- Actually, averaging of output samples leads to removal of noise and more information

Disadvantage:
- Constant/slowly varying input signal is required for (averaging of) multiple similar samples
- Otherwise, dithering increases output noise

Subtractive dithering

- Generate random digital noise using random number generator
- Use DAC to produce random analog noise
- Add this noise
- After ADC, subtract this noise from the digital sig

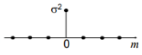
Advantage:
- Doesn't increase output noise unlike simple dithe



Summary:
ERROR ANALYSIS
Rounding: mean = 0, variance = $Q^2/12$
Truncation: mean = $-Q/2$, variance = $Q^2/12$

2) Auto-correlation and power spectral density of zero-mean noise

Or, $R(m) = \sigma^2\,\delta(m)$
where $\delta(m)$ = Kronecker delta function



Summary:

mean of output noise $\eta_v = H(e^{j0})\eta_e$

variance of output noise $\sigma_v^2 = \left(\int_{-\pi}^{\pi} |H(e^{j\omega})|^2 \frac{d\omega}{2\pi}\right)\sigma_e^2$

input and output power spectral density are related by $S_{out}(e^{j\omega}) = |H(e^{j\omega})|^2 S_{in}(e^{j\omega})$

PART 2: sample
d) $50 = f_0 + kf_s = (-25) + 75$
So f_0 = –25, or it yields samples identical to 25 Hz

$$\frac{2f_c + B}{m+1} \le f_s \le \frac{2f_c - B}{m}$$

Peripherals

Amplifier
- The gain is determined to match the input dynamic range to the ADC dynamic range
- For unknown/time-varying input range, use an automatic gain controller (AGC) with time-varying gain

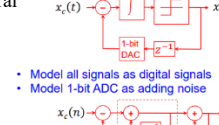
Anti-aliasing filter
- Analog low-pass filter with cut-off frequency B
- Aliasing occurs if analog signal is not bandlimited to $f_s/2$
- Anti-aliasing filter limits the bandwidth of the analog signa

Reconstruction filter
- Also known as anti-imaging filter / smoothing filter
- Smoothens the staircase waveform
- Analog low-pass filter with cut-off frequency B
- Removes spectral copies but retains original spectrum
- In practical ADC, f_s is chosen greater than $2B$ to be able to separate spectral copies by the reconstruction filter

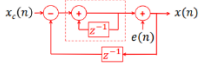
Counter ADC: through DAC compare ref V, one by one, very slow
Successive approximation ADC
Fast, cheap ,ACC
Dual Slope ADC: ACC, slow, but anti-aging
FLASH:fast, low-res more area
Sigma-Delta: low cost, high res, low speed, artifacts.

Analysis of differentiation of the signal:
- Auto-correlation of $x_c(t)$ is $R(\tau) = E\{x_c(t)x_c(t-\tau)\}$
- Variance of $x_c(t)$ is $R(0) = \sigma_x^2$, variance of error is σ_1^2
- $SQNR_1 = 10\log(\sigma_x^2/\sigma_1^2) = 10\log(3\sigma_x^2 2^{2n}/V_{max_x}^2)$
- Differentiation $d(t) = x_c(t) - x_c(t-T)$
- Variance of $d(t)$ is $\sigma_d^2 = E\{[x_c(t)-x_c(t-T)]^2\} = 2R(0)-2R(T$
- Variance of error is σ_2^2, $SQNR_2 = 10\log(\sigma_d^2/\sigma_2^2)$
- If $\sigma_x/V_{max_x} = \sigma_d/V_{max_d}$ then $SQNR_1 = SQNR_2$, both quantizers perform equal, $\sigma_x/\sigma_1 = \sigma_d/\sigma_2$, or $\sigma_2 = \sigma_1(\sigma_d/\sigma_x)$
- If $R(T) > R(0)/2$ then $\sigma_d < \sigma_x$, the variance decreases = range of values decreases = better to quantize $d(t)$
- SQNR $10\log(\sigma_x^2/\sigma^2)$ increases by $10\log(\sigma_x^2/\sigma_d^2)$ dB

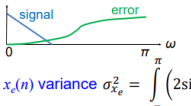
Oversampling
integral



- Model all signals as digital signals
- Model 1-bit ADC as adding noise



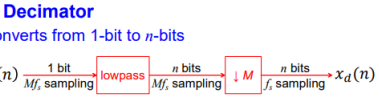
Error is reduced because
- Highpass error $X_e(z) = (1 - z^{-1})E(z)$
- Magnitude of frequency response $|H(e^{j\omega})| = 2\sin\frac{\omega}{2}$
- $e(n)$ power spectral density = N



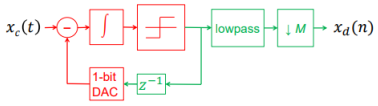
- $x_e(n)$ variance $\sigma_{x_e}^2 = \int_{-\pi}^{\pi} \left(2\sin\frac{\omega}{2}\right)^2 N\frac{d\omega}{2\pi} = 2N$
- $x_e(n)$ variance after lowpass filter $\sigma_{x_e}^2 = \int_{-B}^{B} \left(2\sin\frac{\omega}{2}\right)^2 N\frac{d\omega}{2\pi} \approx \int_{-B}^{B} \omega^2 N\frac{d\omega}{2\pi} = \frac{B^3 N}{3\pi}$
- 2 times sampling = B becomes half = $\sigma_{x_e}^2$ becomes 1/8 = 9dB more SQNR

3) Decimator

Converts from 1-bit to n-bits



4) Block diagram of sigma-delta ADC



first-order sigma-delta ADC (red = analog, green = dig

5) SQNR

Since any quantizer SQNR = 6.02n + .., adding 1 bit resolution = 6dB more SQNR.

For oversampling:
- Each doubling of sampling = 3dB more SQNR = adding ½ bit

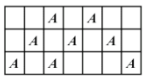
For oversampling and first order noise shaping:
- Each doubling of sampling = 9dB more SQNR = adding 3/2 bits

For oversampling and second order noise shaping:
- Each doubling of sampling = 15dB more SQNR = adding 5/2 bits

Ladder DAC: need good resistor
0 order 1 order interpolation
Reconstruction filter. Sigdlt DAC lowcost

SCHEDULING:

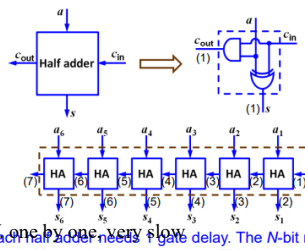
Stage utilization = the fraction of time a given stage is being utilized
Example: stage 1 utilization = 2/7
Number of utilized stage 1 = 2



Maximum achievable (average) throughput ≤ the reciprocal of the maximum number of utilized stages in a single row of the reservation table.

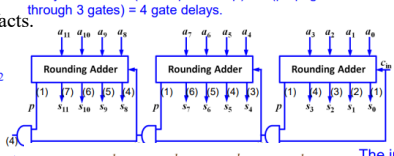
Example: maximum achievable throughput ≤ 1/3

PART 3: ADDER
Rounding adder

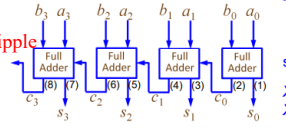


Each half adder needs 1 gate delay. The N-bit rounding adder needs N gate delays to complete. This is because the carry bit propagates from stage to stage.

Therefore, cascading of such blocks is possible.
Carry-out takes 1 (to compute first p) + 3 (propagation through 3 gates) = 4 gate delays.



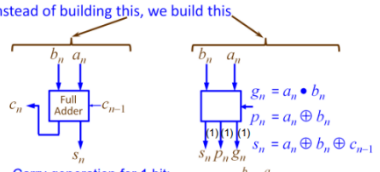
Ripple



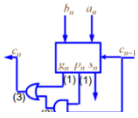
Computing each carry from valid inputs needs 2 gate delays. Therefore, computing c_3 needs 8 gate delays.

In general, an N stage ripple carry adder needs $2N$ gate delays to compute c_{N-1}.

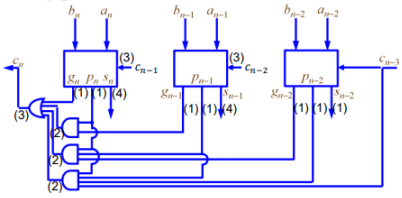
Instead of building this, we build this



Carry generation for 1 bit:
$c_n = g_n + p_n \bullet c_{n-1}$
c_n needs 1 gate delay to compute g_n and p_n, plus 2 gate delays = 3 gate delays.



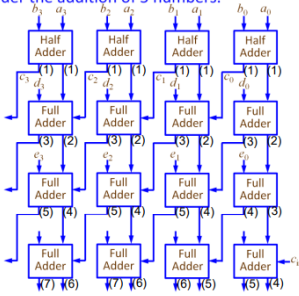
Carry generation for 3 bits:



$c_n = g_n + p_n \bullet g_{n-1} + p_n \bullet p_{n-1} \bullet g_{n-2} + p_n \bullet p_{n-1} \bullet p_{n-2} \bullet c_{n-3}$

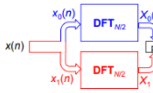
c_n still takes 3 gate delays.
Note that the gate delays will remain 3, 1, 1, 4 for c_n, g_n, p_n, and s_n for any number of bits n.

Carry save adder
Now consider the addition of 5 numbers.



Only the carry bits for the last row ripple "horizontally

Part 4
FFT
Therefore: $X(k) = X_0(k) + e^{-j2\pi k/N} X_1(k)$
This leads to a divide and conquer approach known as the decimation-in-time FFT:



Each $N/2$ point DFT requires $N^2/4$ complex multiply and $N(N-2)/4$ complex add. The "process" requires N complex multiply and N complex add. Overall, the structure needs $N(N+2)/2$ complex multiply and $N^2/2$ complex add.

Recursion: Compute each $N/2$ point DFT using two $N/4$ point DFTs, and so forth. FFT requires about (N/2)log$N$ complex multiply and Nlog$N$ complex add.

Pipelining option 1: insert latches at a feed-forward cutset
2: fine grain.
retiming reduces critical path without increasing latency.

PIPELINED FIR FILTER: rewrite objective function
Sometimes need factorizing as

$$x(n) + x(n-1) + x(n-2) + x(n-3)$$

or

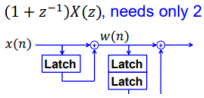
$$\left(1 + z^{-1} + z^{-2} + z^{-3}\right)X(z)$$

needs 3 adders.

However, writing it as

$$(1 + z^{-2})\{(1 + z^{-1})X(z)\} = (1 + z^{-2})W(z)$$

where $W(z) = (1 + z^{-1})X(z)$, needs only 2 adders.



The input 2-phase polyphase components are

$$X_0(z) = \sum x(2n)z^{-n}, X_1(z) = \sum x(2n+1)z^{-n}$$

such that $X(z) = X_0(z^2) + z^{-1}X_1(z^2)$.

$X_0(z)$ is the z-transform of even-numbered inputs $x(2n)$.
$X_1(z)$ is the z-transform of odd-numbered inputs $x(2n+1)$.

Similarly, $Y(z) = Y_0(z^2) + z^{-1}Y_1(z^2)$
and $H(z) = H_0(z^2) + z^{-1}H_1(z^2)$.

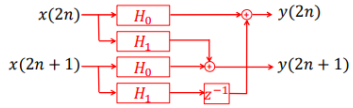
Then $Y(z) = \{X_0(z^2) + z^{-1}X_1(z^2)\}\{H_0(z^2) + z^{-1}H_1(z^2)\}$
$= \{X_0(z^2)H_0(z^2) + z^{-2}X_1(z^2)H_1(z^2)\}$
$+ z^{-1}\{X_0(z^2)H_1(z^2) + X_1(z^2)H_0(z^2)\}$
$= Y_0(z^2) + z^{-1}Y_1(z^2)$

It follows that: $Y_0(z) = X_0(z)H_0(z) + z^{-1}X_1(z)H_1(z)$
$Y_1(z) = X_0(z)H_1(z) + X_1(z)H_0(z)$

In matrix form (leaving z for brevity):

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-1}H_1 \\ H_1 & H_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix}$$

Implementation:



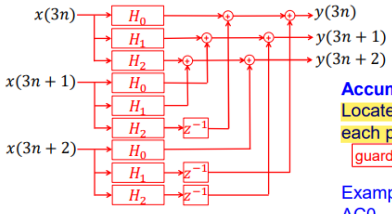
(Note that z^{-1} implies a delay of 2T.)

The polyphase parallel FIR filter requires 2N multiply and 2(N–1) add with double the sampling frequency.
⇒ Strength reduction due to increased throughput.

Alternately, keeping the same sampling frequency, a slower multiplier/adder takes less area.
⇒ Strength reduction due to reduced area.

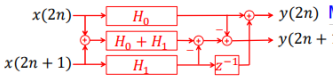
Low-Complexity Parallel FIR Filters

Implementation:



(z^{-1} implies a delay of 3T.)

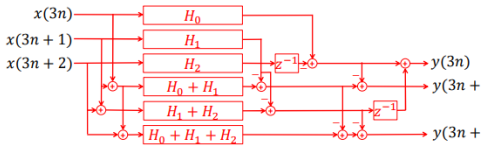
Implementation:



This low-complexity implementation needs only 3 subfilters, each of length N/2.
⇒ It requires 1.5N multiply and 1.5N+1 add.

Other equivalent low-complexity parallel FIR filter structures may be obtained. For example, a structure with 3 subfilters H_0, H_0–H_1, and H_1 may be realized.

Implementation:



This structure needs 6 subfilters, each of length N/3.
⇒ It requires 2N multiply and 2N+4 add.

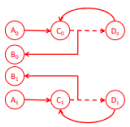
Algorithm for Unfolding

Systematic procedure for unfolding:

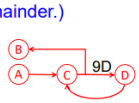
Step 1: For each node U in the original data flow graph, draw J nodes U_0, U_1,…, U_{J-1}.

Original data flow graph:

Replicated data flow graph:

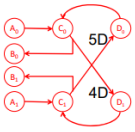


Step 2: For each edge $U \rightarrow V$ with m delays in the original data flow graph, draw J edges $U_i \rightarrow V_{(i+m)\%J}$ with $\left\lfloor \frac{i+m}{J} \right\rfloor$ delays for i = 0, 1,…, J–1.
(% = mod or modulo, find the remainder.)
($\lfloor \bullet \rfloor$ = floor, same as truncation.)

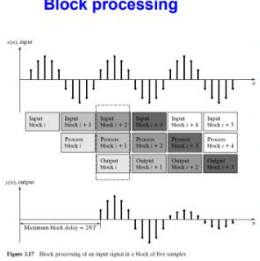


Edges without delay such as $A \rightarrow C$, $D \rightarrow C$ are unchanged.

Edge $C \rightarrow D$ has m = 9 delays.
So, $C_0 \rightarrow D_{(0+9)\%2} = D_1$ has $\lfloor (0+9)/2 \rfloor$ = 4 delays.
$C_1 \rightarrow D_{(1+9)\%2} = D_0$ has $\lfloor (1+9)/2 \rfloor$ = 5 delays.



Block processing



- Collect N samples at a time
- Processing must finish within N sample period, NT
- Normally used in FFT, Compression
- Requires double or triple buffering to perform acquisition and processing
- $T_p < N*T_s$
- A maximum block delay of $2NT_s$

Double buffering

Advantage:
- Can process more data for some given T_s due to the reduced setup time
- Or use a higher F_s compared to the sampling approach

Disadvantage:
- More memory required
- More effort in programming
- Processing latency

C55x Unified Memory Map

Program and data share the same memory

Program memory: 16M x 8 bit, 24-bit address, *(00_0001h) = x

00_0000h	w	
00_0001h	x	
00_0002h	y	
00_0003h	z	
FF_FFFFh	$	7F_FFFFh

00_0000h Data memory: 8M x 16 bit, 23-bit address, *(00_0001h)=yz

segmented into 128 pages (7 msb bits)

page = 64K words (16 lsb bits)

Accumulators: AC0 to AC3
Located in D unit, used for ALU/MAC/shifter output, each portion can be accessed individually or together

guard=bits 39-32	high = bits 31-16	low = bits 15-0

Examples:

AC0	40 bits
HI(AC0)	16 bits (31-16)
LO(AC0)	16 bits (15-0)

MOV AC0, AC1 move 40 bits from AC0 to AC1

Auxiliary registers: AR0 to AR7
Located in A unit, used to address data memory.
ARxH = 7-bit memory page, ARx = 16 lsb bits of address

ARxH = bits 22-16	ARx = bits 15-0

XARx = bits 22-0

Examples:

AR0	16 bits (15-0)
XAR0	23 bits

MOV AR0, AR1 move 16 bits from AR0 to AR1
MOV XAR0, XAR1 move 23 bits from XAR0 to XAR1

MOV HI(AC0), AR0 move 16 high bits of AC0

x < y x bits sign extended to y bits, except when x = 23 bits, x bits zero filled to y bits

MOV AR0, AC0 16 bits of AR0 sign extended to 40 bits

Before		After	
AC0	02_E590_0030h	AC0	00_0000_0024h
XAR0	01_0024h	XAR0	01_0024h

MOV XAR0, AC0 23 bits of XAR0 zero filled to 40 bits

Before		After	
AC0	02_E590_0030h	AC0	00_0001_0024h
XAR0	01_0024h	XAR0	01_0024h

Temporary registers: T0 to T3
Located in A unit, used for multiplicand, shift count, pointer value, transition matrix, etc.

Tx = bits 15-0

Example:
T0 16 bits

Tx works in the exact same way as ARx shown before

Pair of registers: AR0, AR2, AR4, AR6, T0, T2
Refers to 2 registers such as AR0 and AR1

Example:
Pair(AR0) 2 x 16 bits

k4/-k4: 4-bit unsigned constant (with "-" if negative)
K16: 16-bit signed constant
register uses k4, -k4, K16 memory uses K8, K16

k16 absolute addressing:
Denoted by "*"
*abs16(16-bit unsigned constant) = use 7-bit DPH (high part of XDP) + 16-bit constant, to obtain a 23-bit address, then 16-bit data at this address

DPH = bits 22-16	constant = bits 15-0

Example:
*abs16(#0501h) 16-bit data at (DPH)_0501

MOV #256, *abs16(#0501h) move value 256

Before		After	
DPH	00h	DPH	00h
00_0501	FC00h	00_0501	0100h

MOV AC0, *abs16(#0E10h) move 16 bits

Before		After	
AC0	01_4500_0030h	AC0	01_4500_0030h
DPH	00h	DPH	00h
00_0E10	0000h	00_0E10	0030h

Direct Addr use @

@Daddr = Use 7-bit DPH + 16-bit (DP + Doffset), then 16-bit data at this address. Doffset = 7 lsb bits of (Daddr – DP).

	Daddr = bits 15-0
–	DP = bits 15-0
=	neglect | Doffset = bits 6-0

	DP = bits 15-0
+	Doffset = bits 6-0
=	address bits 15-0
DPH = bits 22-16	address bits 22-0

Auxiliary register (AR) indirect addressing:
Denoted by "*"
*ARx = 16-bit data at XARx

XARx = bits 22-0

Example:

*AR0	16-bit data at XAR0
low_byte(*AR0)	8-bit (bit 7-0) data at XAR0
high_byte(*AR0)	8-bit (bit 15-8) data at XAR0
*AR0, *AR1	2x16-bit data, bit 15-0 at XAR0, bit 31-16 at XAR1
dbl(*AR0)	32-bit data. If XAR0 = even, then bit 31-16 at XAR0 and bit 15-0 at XAR0+1. If XAR0 = odd, then bit 31-16 at XAR0 and bit 15-0 at XAR0-1.

MOV *AR0, *AR1, AC0 Move 16-bit data at XAR0 to bits 15-0 of AC0. Move 16-bit data at XAR1, sign extended to 24-bit, to bits 39-16 of AC0.

Before		After	
XAR0	01_0300h	XAR0	01_0300h
XAR1	01_0400h	XAR1	01_0400h
AC0	01_1111_0000h	AC0	00_0300_FF00h
01_0300	FF00h	01_0300	FF00h
01_0400	0300h	01_0400	0300h

MOV AC0, *AR0, *AR1 Move bits 15-0 of AC0 to location at XAR0. Move bits 31-16 of AC0 to location at XAR1.

Before		After	
AC0	01_4500_0030h	AC0	01_4500_0030h
XAR0	00_0200h	XAR0	00_0200h
XAR1	00_0201h	XAR1	00_0201h
00_0200	3400h	00_0200	0030h
00_0201	0FD3h	00_0201	4500h

MOV dbl(*AR0), dbl(*AR1) move 2 consecutive data

Before		After	
XAR0	01_0300h	XAR0	01_0300h
XAR1	01_0400h	XAR1	01_0400h
01_0300	3400h	01_0300	3400h
01_0301	0FD3h	01_0301	0FD3h
01_0400	0000h	01_0400	3400h
01_0401	0000h	01_0401	0FD3h

MOV dbl(*AR0), pair(T0) move bits 31-16 to T0 and bits 15-0 to T1

MOV pair(T0), dbl(*AR0) move T0 to location at XAR0, move T1 to location at XAR0+1

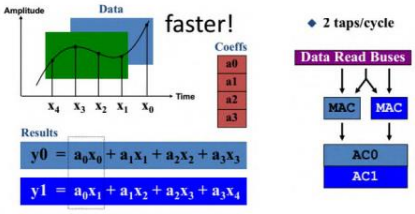
1) **MAC** with registers: m1 = ACx, m2 = Ty
ACx: only bits 32-16 are multiplied
Ty: sign extended to 17 bits

2) **MACK** with value: m1 = Tx, m2 = K8/K16
K8/K16: 8/16-bit signed value, sign extended to 17 bits

3) **MACM** with memory: m1 = memory, m2 = ACx/Tx/memory
memory: sign extended to 17 bits

FIR Filtering in C55x

User-defined parallelism example: **MOV || MOV**
MOV register, register || MOV register, register



C55x: MAC *AR2+, *CDP+, AC0 :: MAC *AR3+, *CDP+, AC1

Achieves 2 taps/cycle using two MACs in parallel

A linear/circular configuration bit in status register controls whether a pointer is linear or circular.

A circular buffer consists of:
- One buffer start address register
- One buffer size register
- Pointer offset stored in ARx or CDP

It is possible to have up to 3 simultaneous circular buffers of different sizes.

Overflow
- In D unit ALU/MAC, overflow is detected typically at bit 31
- Accumulator overflow status bit (ACOVx) is set to 1

Saturation
During overflow, the result may be saturated.

- If saturation mode bit for D unit (SATD) is 1, then positive result is typically saturated to 00_7FFF_FFFFh and negative result is saturated to FF_8000_0000h
- If saturation mode bit for A unit (SATA) is 1, then positive saturated to 7FFFh and negative to 8000h
- If saturation on multiplication bit (SMUL) is 1, then same as SATD for multiplication result

If fractional mode status bit (FRCT) is 1, products are automatically shifted left by 1 bit.