Programmable Digital Signal Processors: Part1

Prof. Wonyong Sung

School of Electrical Engineering
Seoul National University

Contents

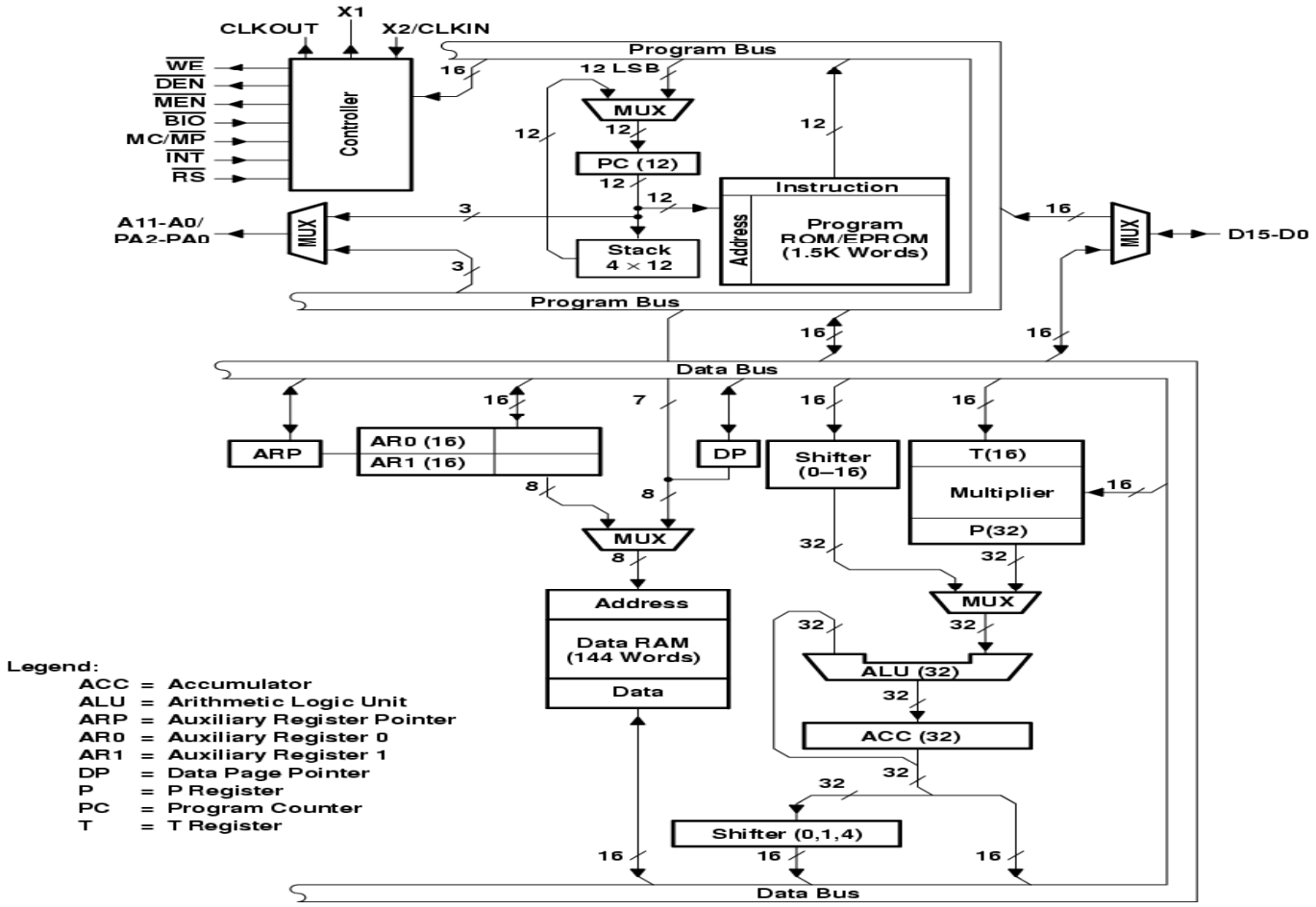
1. **Introduction**
2. **Architectural features for programmable DSP**
   - Fixed-point and floating-point ALU
   - Address generation unit
   - Internal memory blocks and buses
   - Program control unit and pipelining
3. **FIR filtering with C5x**
   - Scaling methods
   - RPT instruction
   - FIR filtering
   - Adaptive filtering
4. **Programmable DSP vs RISC CPU**

1. Introduction

❖ What is programmable DSP?

- Microprocessor or microcomputer for real-time implementation of digital signal processing algorithms
  - Control, audio -> video
- Architecture for high throughput low-power signal processing
  - 10 MOPS – 200 MOS (several GOPS for VLIW DSP)
- Supports special purpose instructions for filtering, FFT, communication and multimedia (2nd generation)
- Program (not HW) based implementations ->
  - good for small quantity production
  - Good for flexible and post-modifying implementations

TMS32010 BLOCK DIAGRAM

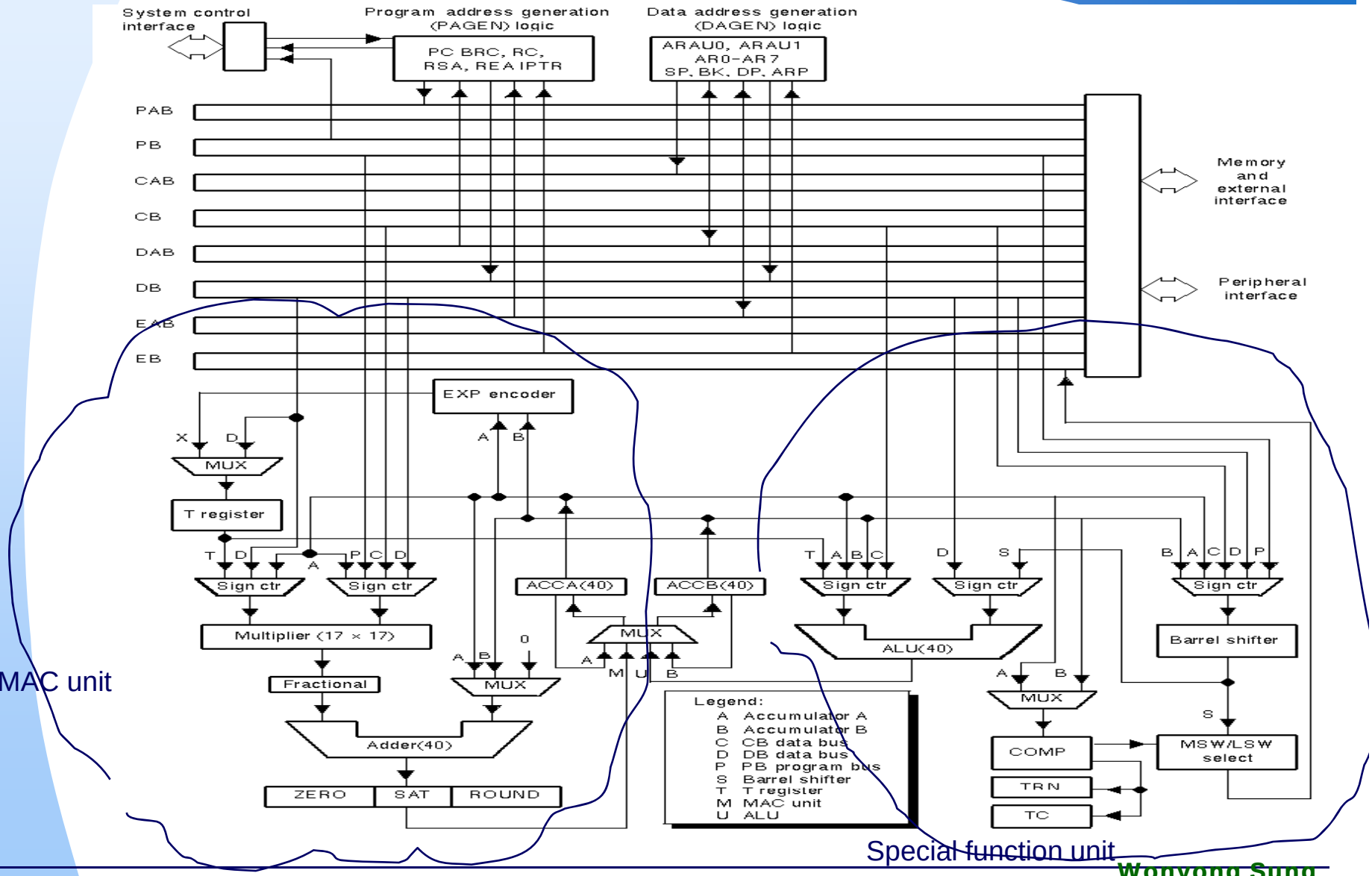


Legend:

ACC	=	Accumulator
ALU	=	Arithmetic Logic Unit
ARP	=	Auxiliary Register Pointer
AR0	=	Auxiliary Register 0
AR1	=	Auxiliary Register 1
DP	=	Data Page Pointer
P	=	P Register
PC	=	Program Counter
T	=	T Register

TMS320C54x Internal Block Diagram

TI Document



MAC unit

Special function unit

DSP history

- **1980: NEC 7720 – DSP supporting 1 cycle MAC**
- **1982: TI TMS32010 (-> 2x, 25, 50, 54)**
- **1985: NEC 77230 floating-point 32bit DSP**
- **1987: Motorola DSP56000**

- **Early applications of prog. DSP**
  - Telephony modem,
  - LPC vocoder, voice echo canceller

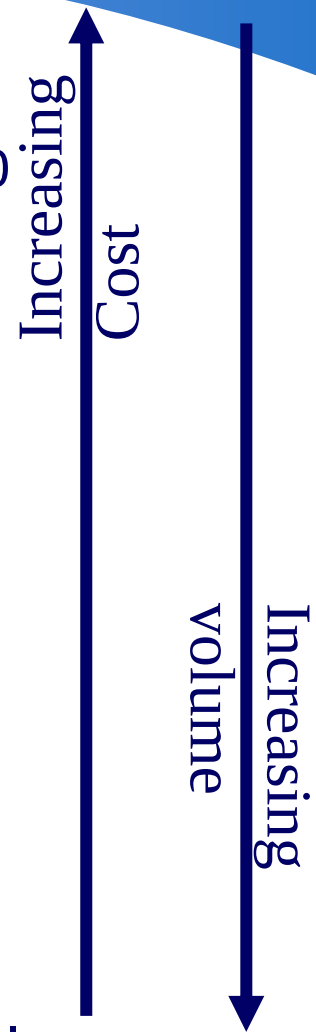
- **Current applications of programmable DSP**
  - CELP vocoder for cellular phones (TI for GSM phones)
  - Viterbi decoder for error correction, speech recognition
  - MP3 player (Sigmatel, DSP56000 + internal memory + codec + DC-DC converter + USB)
  - Audio decoder for DVD, HDTV
  - Base station for cellular phones (C6x VLIW)

TI DSP history and family

- **C10: 2 cycle for FIR filtering (obsolete)**
- **C25, C50: 1 multiply and 1 adder**
  - Intended for single cycle FIR filtering
- **C3x: floating-point DSP, good C compiler support (obsolete)**
- **C4x: multiprocessor extensible floating-point DSP (obsolete)**
- **C54x: 1 multiply and 2 adder (2nd generation)**
  - Intended for two tap processing of linear phase FIR filtering
  - Viterbi decoding, FFT
  - Currently cash cow (because of GSM)
- **C55x: 2 MAC (multiply and accumulation) inside**
  - C54x + alpha
- **C8x: multiprocessor DSP (4 fixed-point DSP + 1 RISC inside) (obsolete)**
  - Intended for image processing, mpeg applications
  - discontinued because of difficulty of programming
- **C6x: VLIW DSP, good compiler support (3rd generation)**
  - C62 integer, C64 packed data, C67 floating-point
  - Intended for ADSL modems but too expensive!
  - Used for imaging, and many high performance applications

DSP family and applications

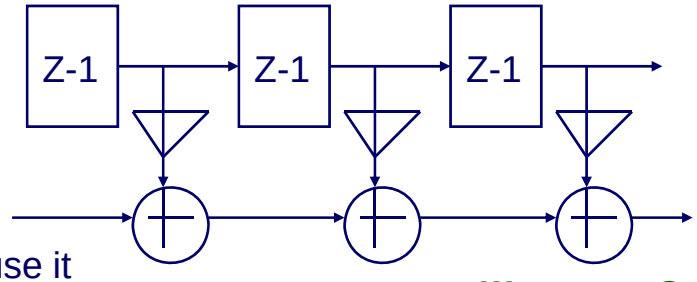
- **High-end**
  - Wireless Base Station - TMS320C6000
  - Cable modem
  - gateways
- **Mid-end**
  - Cellular phone - TMS320C540
  - Fax/ voice server
- **Low end**
  - Storage products - TMS320C27
  - Digital camera - TMS320C5000
  - Portable phones
  - Wireless headsets
  - Consumer audio
  - Automobiles, toasters, thermostats, ...



Programmable DSP Architecture

- **Initially motivated for efficient filtering (FIR and IIR)**
  - FIR filtering
    - It takes a total of 5 ~10 cycles in RISC processors, but just one or less cycle in programmable DSP's. Ex: C1x, C2x
  - For 1 tap/cycle FIR filtering
    - Two data read (one coefficient, one data)
    - Two next address calculation (simple address increase)
    - Multiply of coefficient and data
    - Accumulation of the product
    - Loop end control (to see if end of the filter tap)
  - Later improved to support some other DSP algorithms efficiently (such as Viterbi, FFT, LMS adaptive filtering)
    - Ex:. C54x, C55x

```
*output = 0;
for (j=0; j<NTAPS; j++)
    *output += (*data--)*(*coef++);
```



*In C, pointer based implementation is better (faster) because it uses AGU.

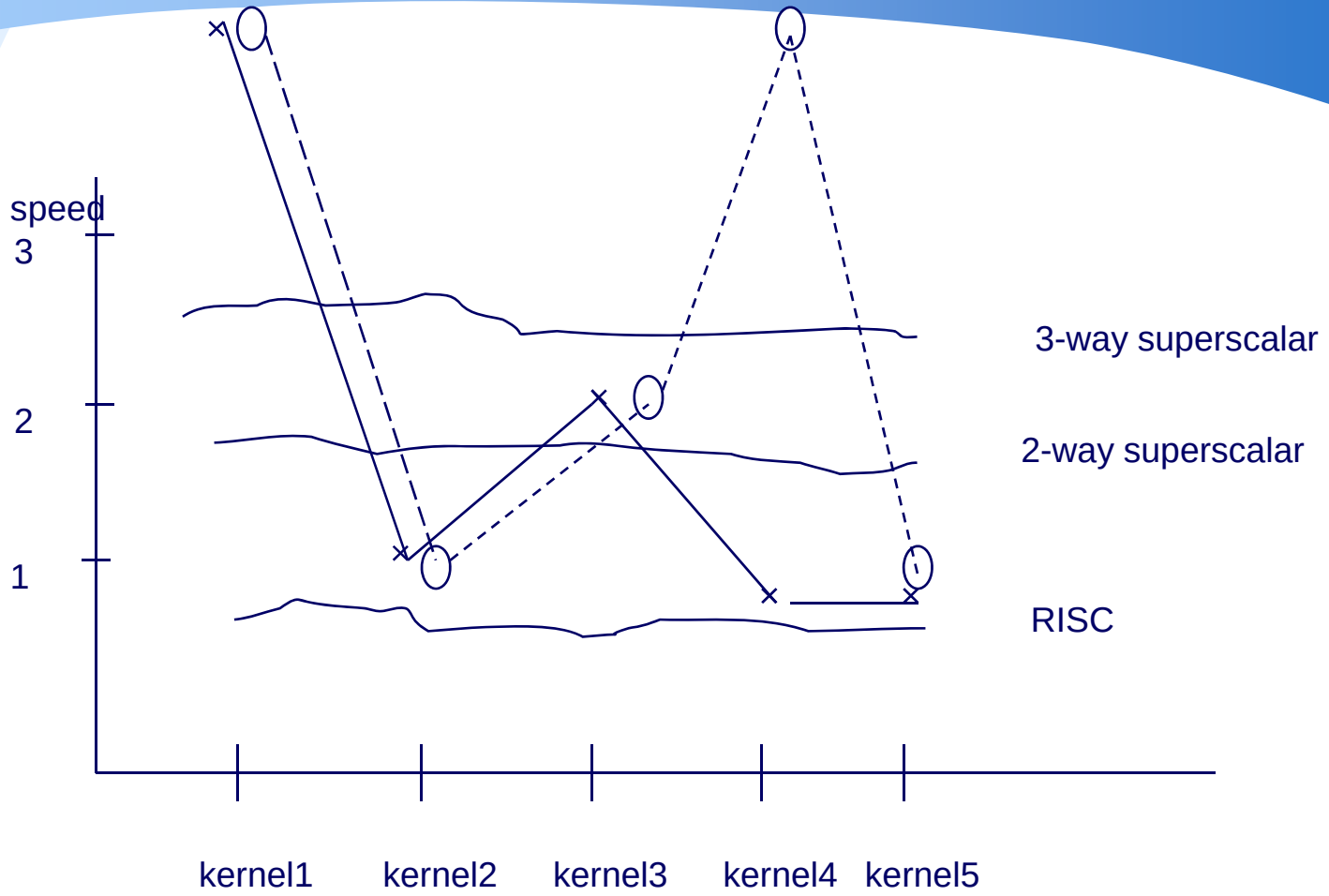
How to conduct 1tap/cycle FIR filtering?

- **Single cycle instruction execution (pipelining)**
  - Now employed to RISC processors
- **Parallel operations in one cycle (micro-parallelism)**
  - Execution unit:
    - Simultaneous multiply, add, shift operations in one cycle
    - Simultaneous next address generations (two address generation units)
  - Memory and registers:
    - Multiple memory blocks for multiple data read in one cycle
    - Several special purpose registers for parallel operation (not good for compilers)
  - No loop overhead (HW repeat unit)
  - Bus
    - Separate instruction and data buses (Harvard architecture)
    - Now employ more than one data bus (3 for C54x)
  - (Special functional units)
    - Compare, select, unit
- **Dense instruction encoding**
  - All of the parallel operations are encoded in one instruction word.
    - MACD
- **Programmable DSP is a CISC machine tailored to DSP applications.**

Weakness of programmable DSP's

- **Addressing range is usually small (about 64Kwords~1M words) because the processors are intended for running one or two real-time applications continuously.**
  - Small addressing range is efficient in terms of memory usage, power consumption
- **Not efficient for complex control oriented programs**
  - Pipelining and micro-parallelism utilizing several hardware simultaneously are not generally applicable
- **The average speed-up (versus naïve RISC CPU) is very much program dependent while superscalar architecture shows rather consistent speedup.**
- **C programming is not well supported.**

Programmable DSP vs Superscalar RISC



2. Architectural features for programmable DSP

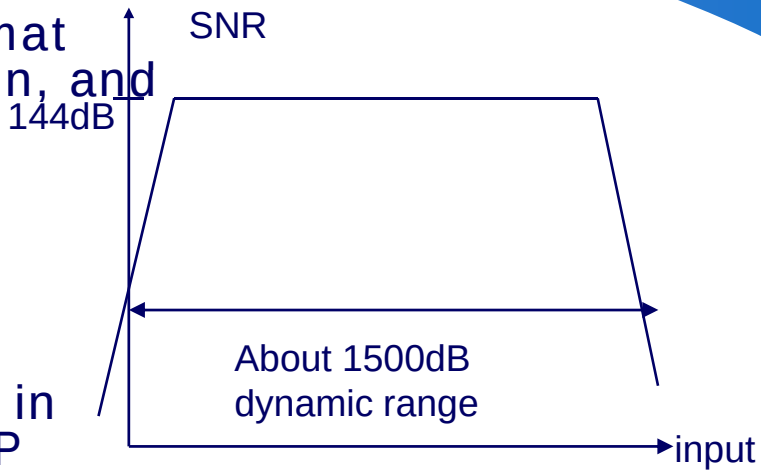
- **Fixed-point or Floating-point ALU**
  - Parallel multiplier, adder, special function units
- **Address Calculation Unit**
  - Separate address calculation units (usually two)
  - Separate address registers
- **Multiple Internal Memory Blocks and Buses**
  - Multiple memory blocks and buses for fast data transfer
  - Support the demand ratio more than 1
    *demand ratio: the number of memory reads/writes in one cycle
- **Program Control Unit**
  - Hardware repeat unit for no-overhead loop control
  - HW stack
  - Highly pipelined operation
- **Specialized IO – such as ADC/DAC**
- **(Special functional units and supporting instructions)**
  - For second generation
  - C54x

a) Floating-point vs fixed-point DSP

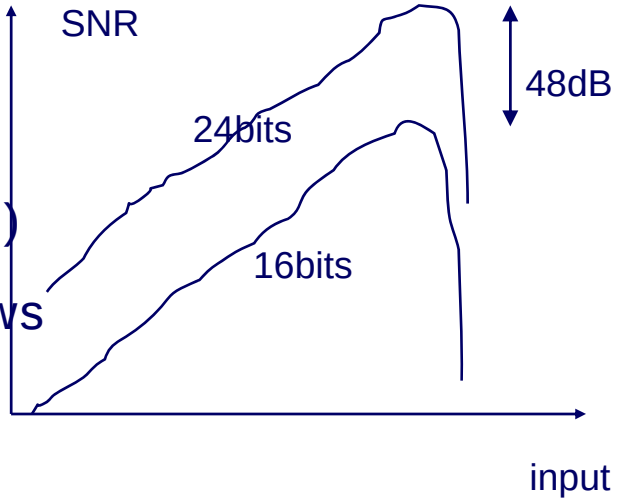
- **Floating-point DSP**
  - Floating-point arithmetic unit, 32bit data width
  - Efficient C Compiler => Fast development, but larger power and cost
  - TI : TMS320**C30**, C40, **C67x** (VLIW floating-point DSP)
  - Motorola : DSP96k, AT&T : DSP32xx, Analog Device : ADSP210x0, NEC μPD77230 DSP
- **Fixed-point DSP**
  - Low cost, low power
  - Need scaling, assembly programming -> good for mass production (cellular phones, MP3 decoders)
  - Many are highly integrated (ADC, DAC, large on-chip memory, on-chip IO)
  - TI : TMS320C25, C50, **C54x, C55x, C62x, C64x** (MMX VLIW)
  - Motorola : DSP56k Series, AT&T : DSP16xx series
  - Zoran : ZR38k series, NEC μPD7720

Floating-point and fixed-point datapath

- **Floating-point**
  - Usually employ IEEE standard format 32bit (24bit mantissa including sign, and 8 bit exponent)
  - $2^{-127} <= |N| <= 2x2^{127}$
    $5.9x10^{-39} <= |N| <= 3.4x10^{38}$
  - Dynamic range about 1500 dB
  - No need to worry about scaling
  - X2 or x4 higher in cost and slower in speed compared to fixed-point DSP
  - Easier development

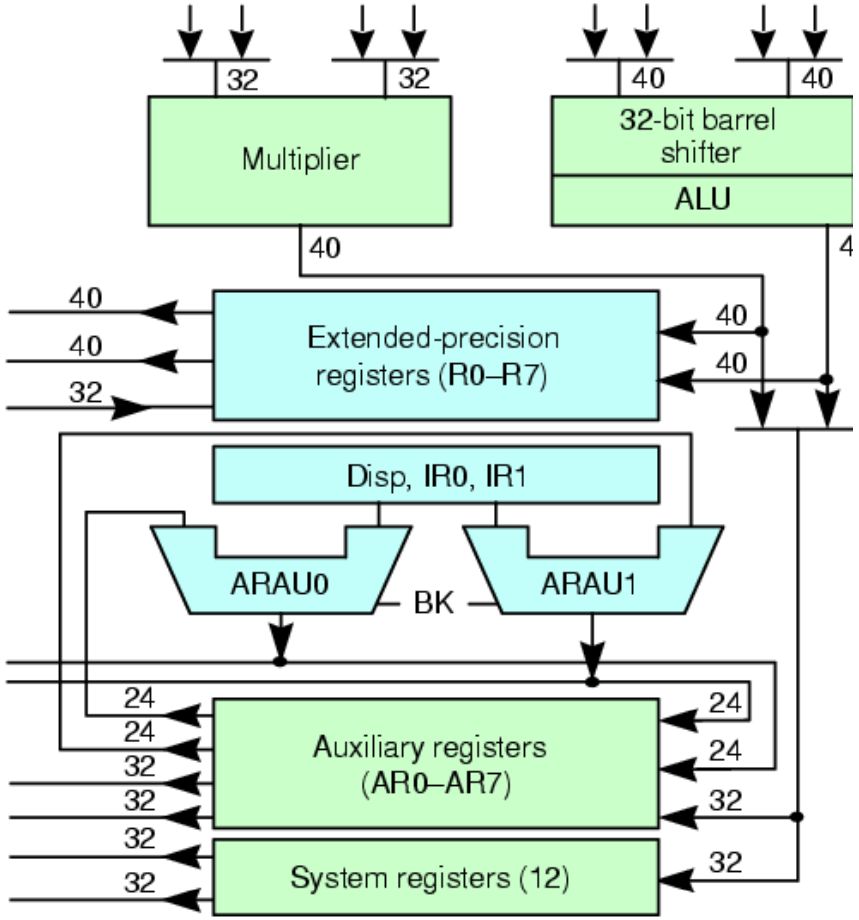


- **Fixed-point (16~24bits)**
  - Multiplier, adder, and barrel shifters
  - Employ double precision accumulator(s) (32bit), sometimes with guard bits (extended precision to prevent overflows while accumulating)
  - Should concern about scaling (determining the number of shifts)

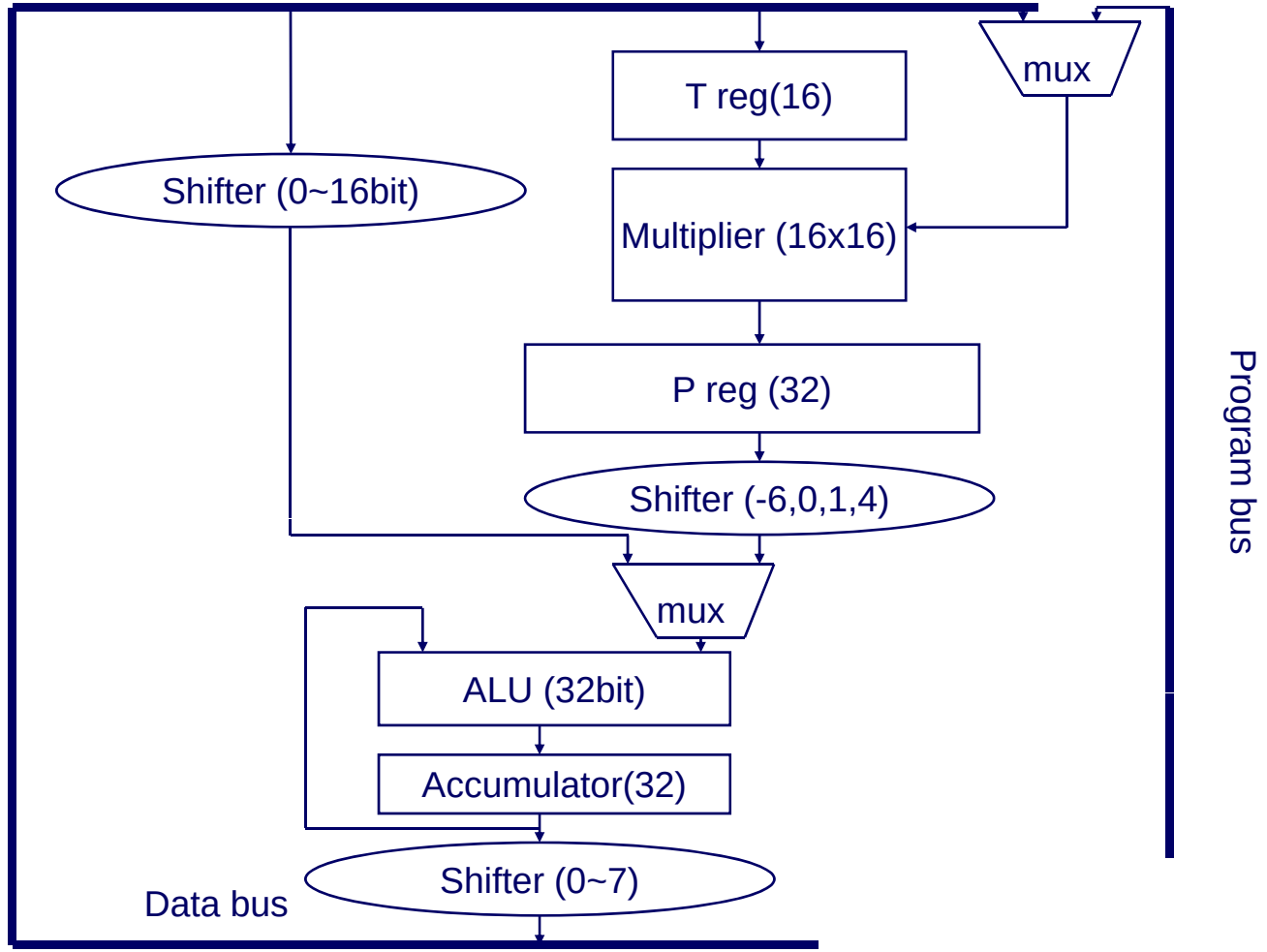


TMS320C3x CPU BLOCK DIAGRAM

TI Document

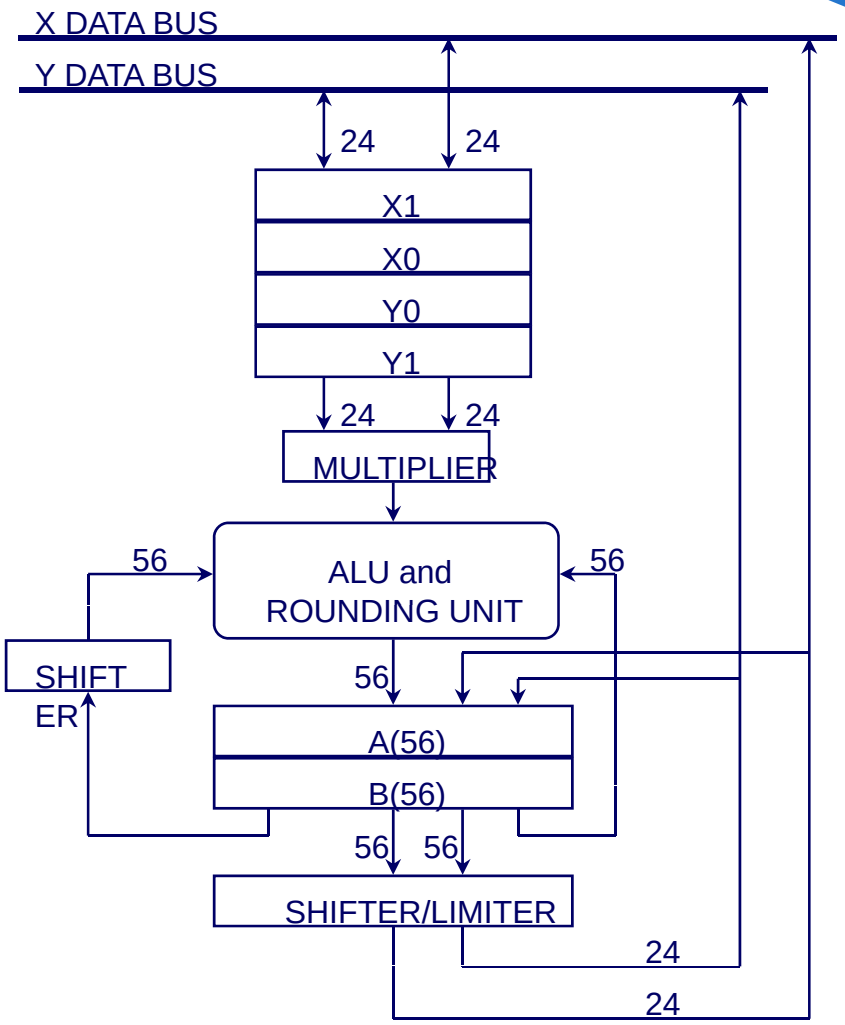


C25 Datapath



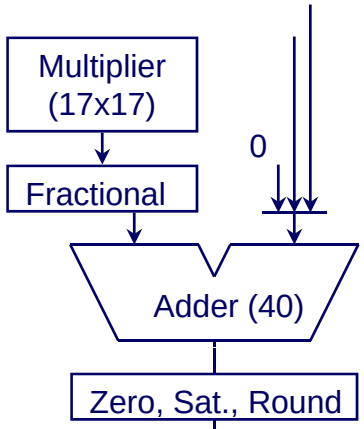
DSP 56000

- **24bit DSP : Good for audio**
- **ALU,Acc : 56 bit(8 guard bits)**
- **Integrated MAC : 1cycle latency**
  - Addition and multiplication, MAC use the same datapath
  - Multiplier conducts the function of pre-scaler
- **Only post-scaler**
  - no need of s0 because of guard bits
  - for addition, multiplier becomes the shifter

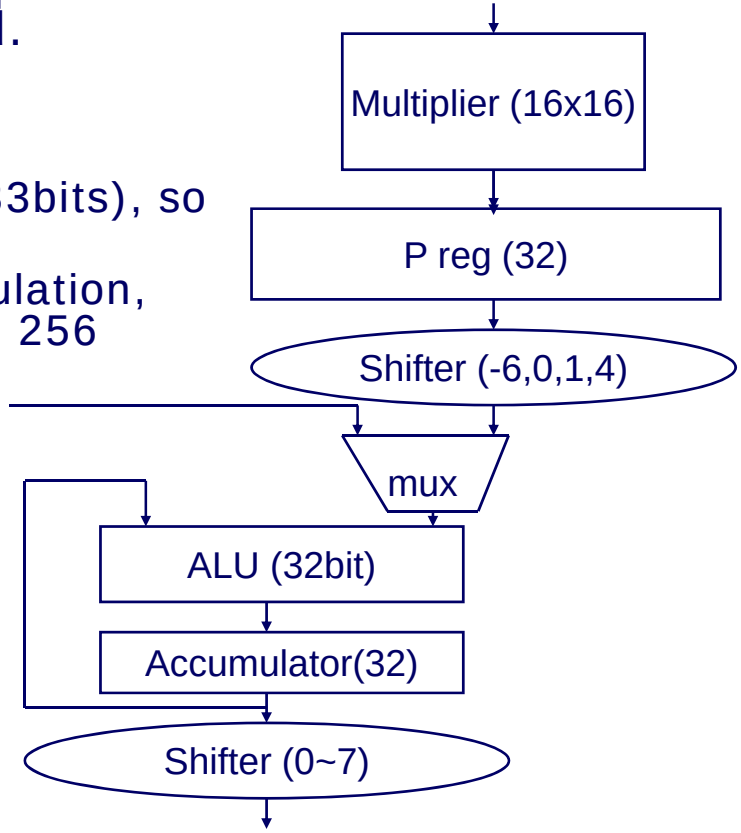


Scaling: shift right and guard bits

- **Shift right (-6bit in C2x, C5x)**
  - Multiplication results and accumulator are all 32bits
  - So, before accumulation of multiplied results, shift 6bit down is needed.
- **Guard bits (8bit in C54, C55)**
  - Total of 40bit for ALU and ACC
    - Multiplication results are 32bits(33bits), so upper 8(7) bits are guard-bits.
    - No need to shift down for accumulation, and no overflow guaranteed upto 256 repeated accumulation
- **Saturation or overflow mode**



C54x



C5x

Pipelining between multiplier and ALU

- **Pipelining: P reg (for C2x, C5x)**
  - Helps for speeding up the hardware
  - It complicates the programming.
- **No pipelining: C54x, Motorola 56000**
  - Easier programming

- **Ex1: C2x, C5x FIR filter**

```
zpr; zero product register
rpt  #FILT_ORDER-1
macd  #coef, *- ;ar3=&z[FILT_ORDER]
apac ; Add the P register to ACC
```

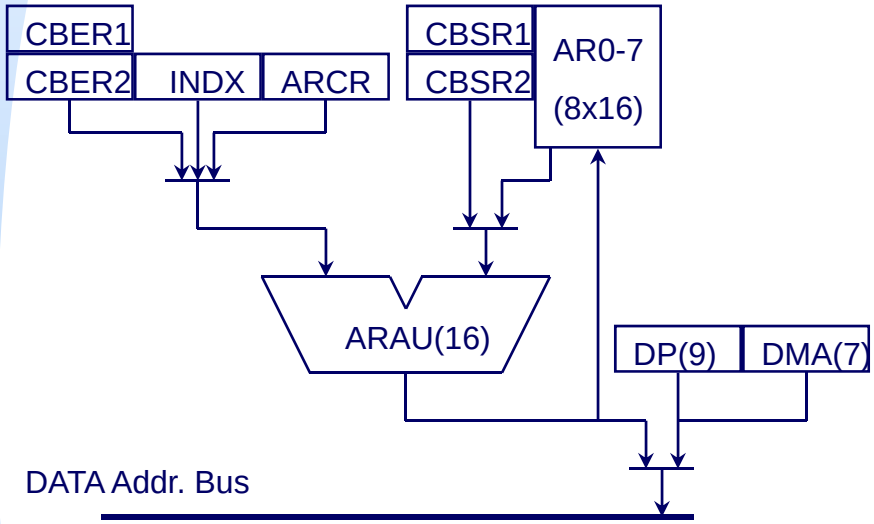
- **Ex2: C54x FIR**

```
rpt #FILT_ORDER-1
mac *AR2+, *AR3+, A
```

b) Address generation units and address registers

- Address calculation for next data accesess – AGU(Addr. Generation Unit)
- C3x: two AGU's, 8 AR's
- C2x,C5x : 1AGU, 8AR(Addr. Register) but PC (Program Counter) is used for incrementer for RPTK instructions
- DSP56k : 2AGU,24AR
- Circular addressing : wrap around of addresses
  - Cf) FIR filtering
- FFT – bit reverse addressing
- Offset addressing : index register 이용

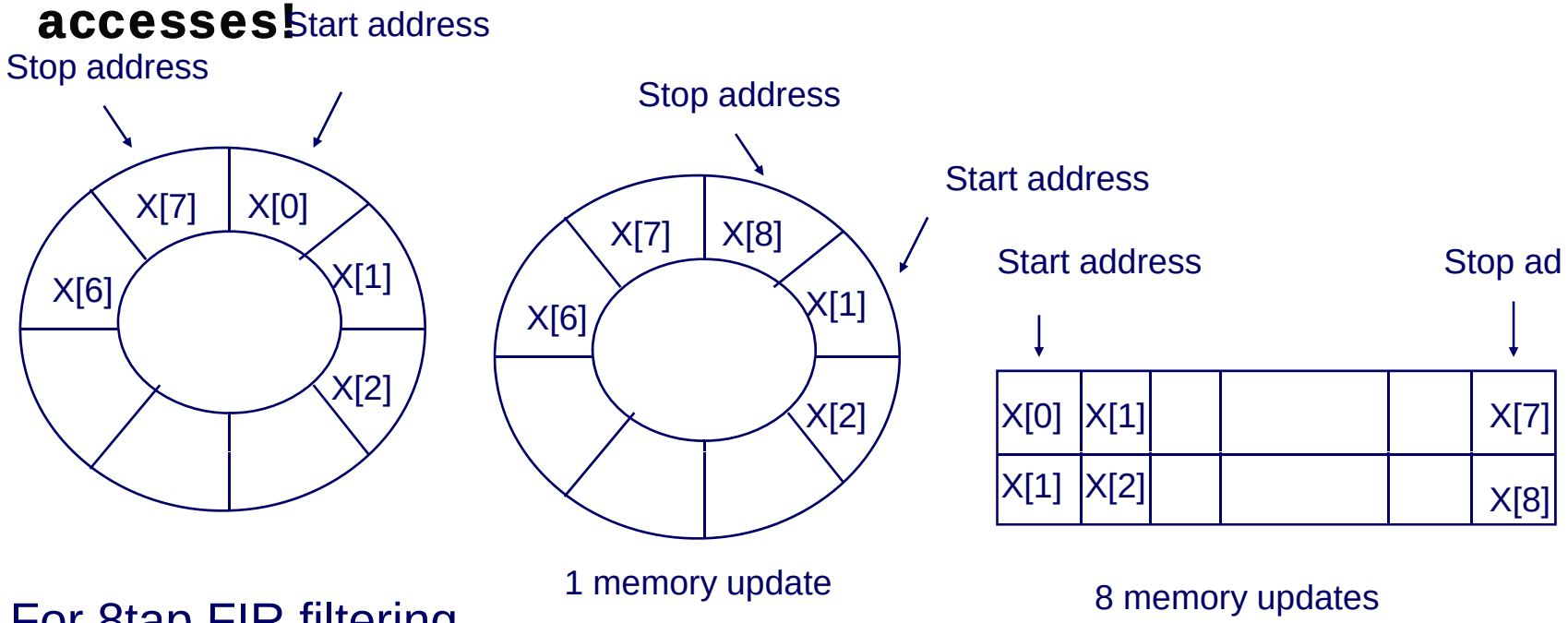
Address generation unit of TMS320C5x



- CBER : Circular Buffer End addr. Reg.
- CBSR : Circular Buffer Start addr. Reg.
- AR(0-7) : pointer register
- ARAU : Address calculation
- INDX : indx reg. index value for ARAU (increment/decrement)

Circular and linear addressing for FIR filtering

- **Assumes a circular memory model, while the start address advances by one at each iteration and the data at the end is replaced by new one.**
- **Other method: move data by one after read (delayed write) C25 – this causes more memory accesses!**

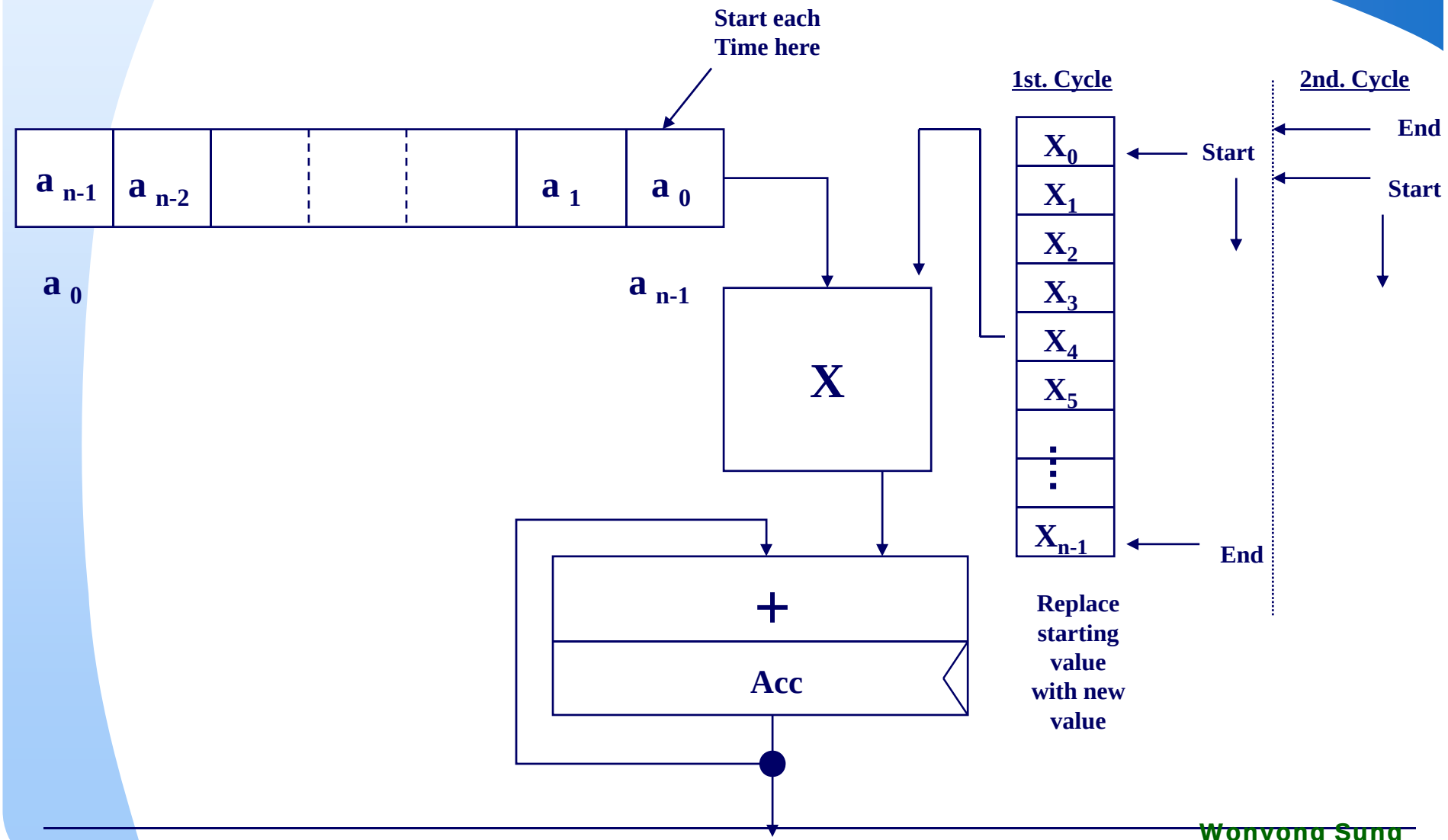


1 memory update

8 memory updates

For 8tap FIR filtering

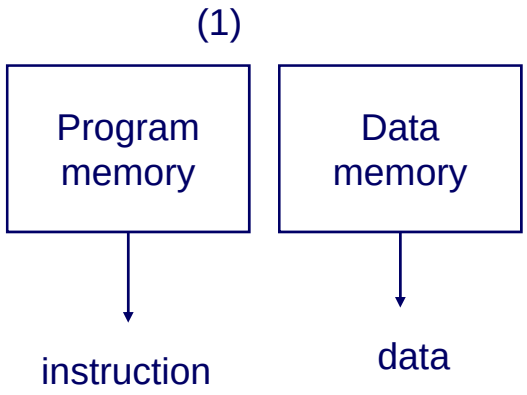
Digital FIR Filter Implementation (Uniprocessor-Circular Buffer)



c) Internal memory blocks and buses

- **Harvard architecture: Separate program and data buses**
  - Many advanced DSP's have multiple data buses
- **Direct memory accesses are supported in DSP**
  - Why? DSP application needs frequent data memory accesses (Instruction profiling)
- **Demand ratio : max memory accesses/cycle**
  - For single cycle FIR filter: need demand ratio of two
  - For linear phase FIR filter two tap per cycle (C54x): need demand ratio of three

Harvard architecture and variations

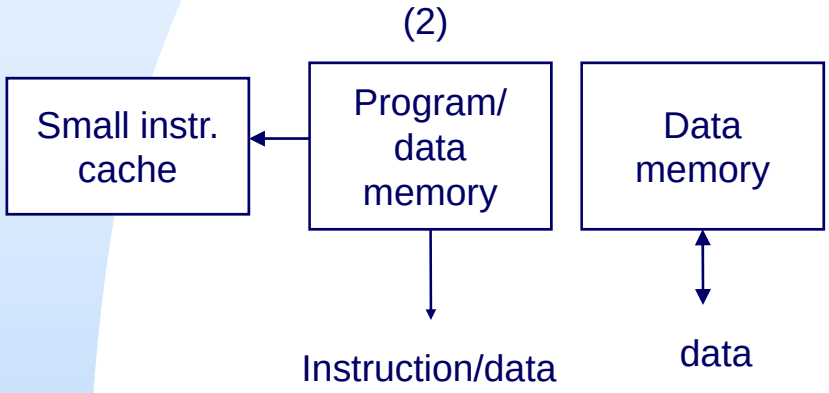


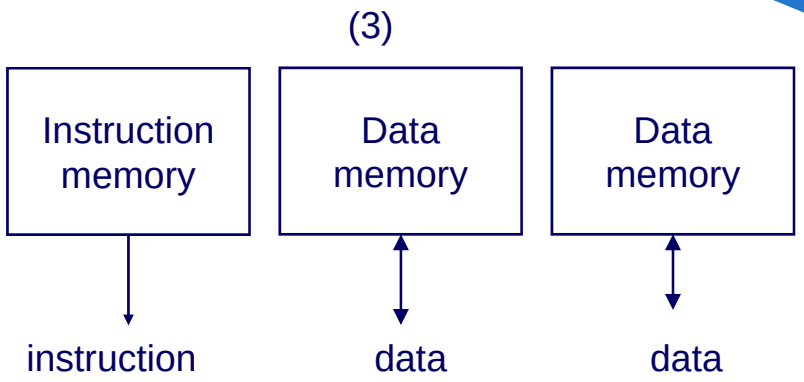
TMS32010 FIR Filter

```
LT COEF(N)
MPY DATA(N)
LTD COEF(N-1)
MPY DATA(N-1)
 ----
LTD COEF(1)
MPY DATA(1)
APAC
SACH  RESULT, 1
```

* It takes two instruction cycles per tap!

Harvard architecture and variations



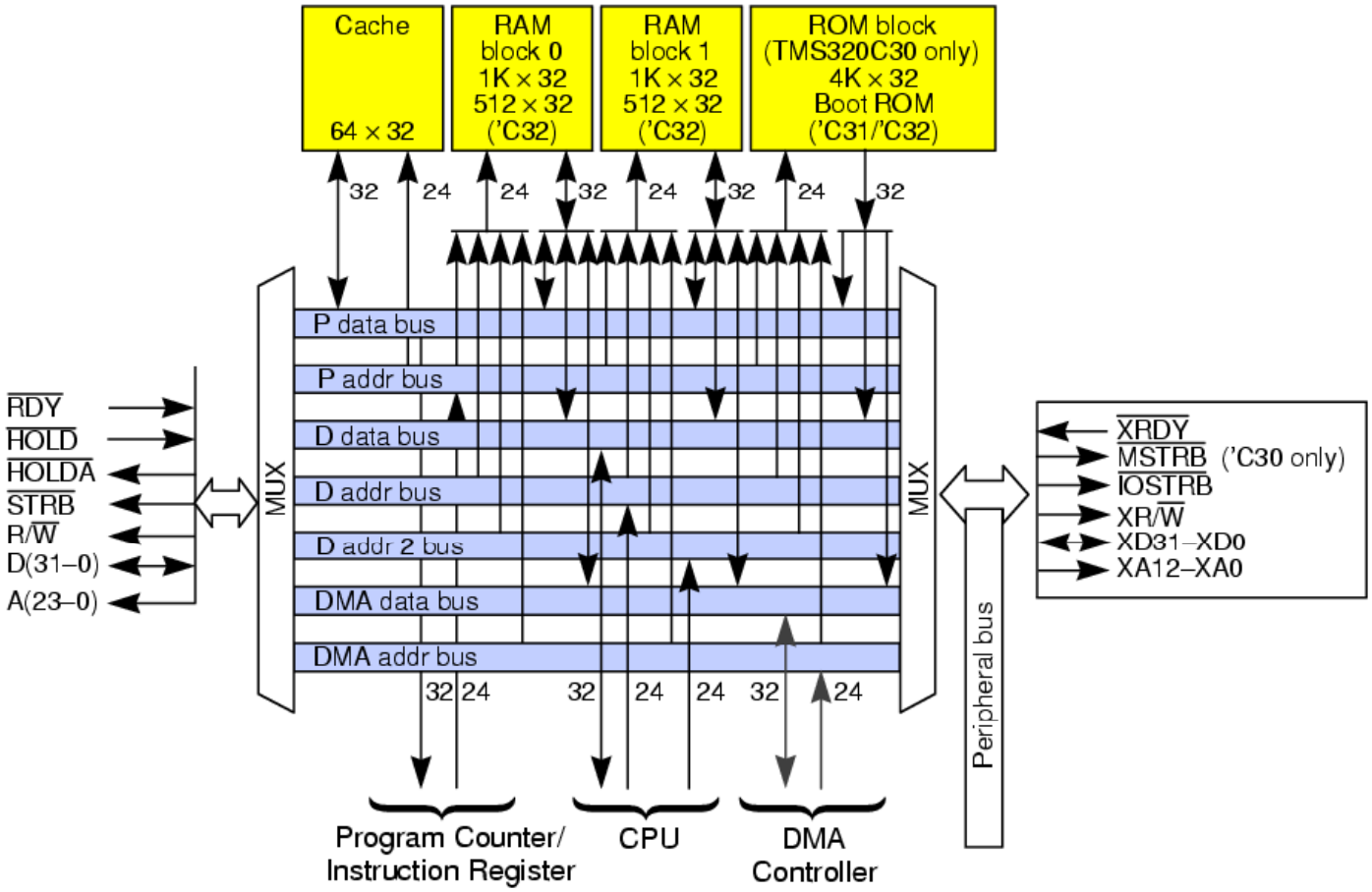


TMS 320C3x

Needs two cycles for FIR one tap
(TMS 320C20, 25)

RPTK constant
MACD address, *-
//* the address is loaded into PC
And PC is used for auto-increment register

TMS320C3x memory block diagram



d) Program control

- **Hardware loop: eliminates the overhead of loop end test (loop index decrease and jmp nz…)**
  - Single cycle loop
  - Multi cycle loop (repeat block)
- **Hardware stack**
- **Efficient interrupt service**
  - No interrupt possible when single cycle loop (RPT)
  - C5x: automatic register saving using shadow registers

Processor operation and pipelining

- **Operations for 1 cycle MAC**
  - Instruction read and decode
  - Operands read
  - Multiply and accumulation,
  - Next address calculation
- **Reservation Table for C5x**

cycle	1	2	3	4	5	6
P.Mem	Fetch1	Fetch2	Fetch3	Fetch4	Fetch5	Fetch6
D.Unit		Dec1	Dec2	Dec3	Dec4	Dec5
AGU		Dec1	Dec2	Dec3	Dec4	Dec5
D.Mem			Op1	Op2	Op3	Op4
Mult				Ex1	Ex2	Ex3
ALU				Ex1	Ex2	Ex3

Pipeline hazards

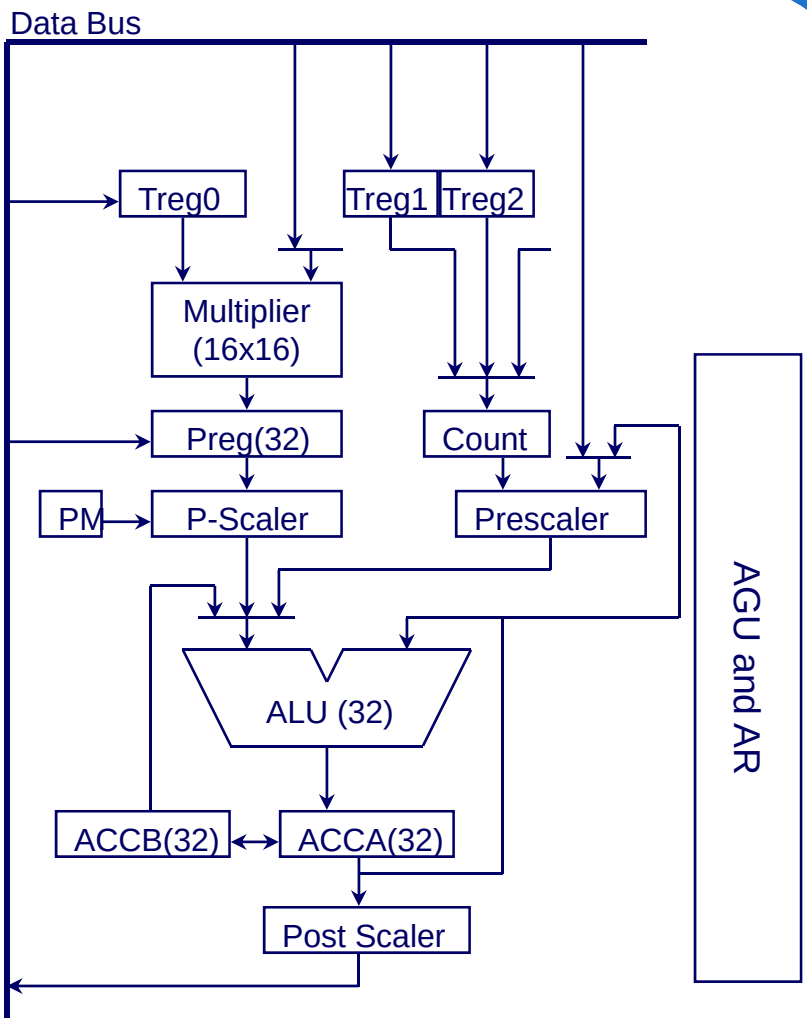
- **Data, branch, resource hazards**
- **HW interlocking for branch and resource hazards**
  - Lose one cycle (or more?)
  - Example: when the instruction2 has a long immediate
- **No interlocking for data hazards – should be careful**
  - Ex: C5x AR registers can be updated by the ALU in the datapath. In this case, the updated value cannot be used at the next instruction cycle. Needs two delay slots. If done using AGU, it is no problem since it is conducted at the DEC stage.

HW interlocking

cycle	1	2	3	4	5	6
P.Mem	Fetch1	Fetch2	Fetch3	Fetch4	Fetch5	Fetch6
D.Unit		Dec1	Dec2		Dec3	Dec4
AGU		Dec1	Dec2		Dec3	Dec4
D.Mem			Op1	Op2		Op3
Mult				Ex1	Ex2	
ALU				Ex1	Ex2	

3. FIR Filtering with C5x

- 1st generation programmable DSP for FIR filtering
  - (2nd generation – from C54x)
- 1 MAC, 1 AGU, … hardware efficient architecture
- Data in, out 16 bit, Acc 32bit
- 3 shifters: P, Prescaler, Post scaler
- Special regs: Preg, Treg



TMS320C5x architecture – datapath

Multiplier

Input : Treg0 and (data memory or prog. memory)

Output : Preg.: 16x16 = 32 bit register

P-scaler (Multiplier output scaler) : 0, 1, 4, -6 bit shifter

-6 : this is used for preventing overflows in repeated accumulations.

Controlled by PM(Product mode) register (Use SPM instruction)

ALU

32 bit Addition/Subtraction/Logical

Input : (ACCB, Preg., Memory) and ACC

Output : ACC(32 bit)

Pre-scaler : 0-16 bit barrel shifter. The number of shifts is given by the instructions or dynamically given by Teg1(2).

Post-scaler : Use for extraction 16 bits from the 32bit ACC. Note that the memory is 16bit in C5x. 0 – 7 bit shift of ACCH(Upper 16 bit of ACC) or ACCL.

FIR filtering

$$y[n] = x[n] + \sum_{k=1}^{N-1} a[k] \cdot x[n-k]$$

- C program based implementation

```
for (I=0;I<FRAME;++I) {
        z[0] = x[I];
        acc = 0;
        for (j=N;j>0;--j){
                acc += z[j] *b[j]; MAC
                z[j] = z[j-1];  input data move
        }
        y[I] = acc + x[I];
}
```

x[i] : input, z[j] : delayed data, b[j] : filter coefficients, y[i] : output

Programming strategies for FIR

- **It's most important to optimize the inner most loop**
  - Just next to RPT
- **Block repeat loop calculates one sample output for FIR filtering**
- **Circular buffer programming (end, start, control): cber, cbsr, cbcr**
- **Block repeat count register: brcr**
- **macd: multiplication of current data and coefficient, accumulation of previous product, z[k] is one sample delayed**
  - During the macd, the program bus is used for fetching coefficient
- **apac is needed for concluding the final accumulation**

Innter most MACD operation

```
rpt      #FILT_ORDER-1
macd  #coef, *-          ;ar3=&z[FILT_ORDER]
```

```
for (j=N; j>0; --j){
        acc += z[j] *b[j];
        z[j] = z[j-1];
}
```

- Fetch two operands : z[j], b[j]
- Multiply and accumulate : *acc += z[j] *b[j];*
- data move : *z[j] = z[j-1];*
- modify ar registers : pointer decrease for *z[j], b[j]*

Scaling for FIR filtering in C5x

- **We may use generalized fixed-point format**
- **Scaling for FIR filtering**

```
for(I=0;I<FRAME;++I_){
        z[0] = x[I];
        acc = 0;
        for (j=N;j>0;--j){
                acc += (z[j]*b[j]) >> s0;
                z[j] = z[j-1];
        }
        y[I] = Upper16((acc + (x[I] <<s1))<<s2);
}
```

- s0 is needed for multiplier output scaling
- s1 is needed for direct input scaling
- s2 is needed for output scaling

- **S0 : Preg (mult output) shift (-4, 0, 1, 6)**
- **S1 : pre-scaler shift (barrel shifter)**
- **S2 : post-scaler shift**
- **s0,s1,s2 determination strategies in C5x**

  - IWL determination
  - (x[I] = 13, b[I] = 2, y[I] : 13, acc :17)IWL
  - Determine the iwl of the product
  - (iwl(z[j]*b[j]) = iwl(z[j])+iwl(b[j]) = 13 + 2 + 1 = 16
  - S0=iwl(acc) – iwl(z[I]*b[j])
  - (s0는 –6,0,1,4 so in order to prevent overflow s0 needs to be 6, iwl(acc) becomes 16+6=22)
  - S1 = iwl(acc) – iwl(x[I]) =9
  - S2 = iwl(z[I]) – (iwl(acc)-16)=7

FIR filtering (assembly language)

Circular buffer

block repeat

single repeat

```
        spm   0
        mar   *,ar5                    ; spm = mult output shift mode
        splk  #z,cber1      ;dp=0      ; splk = store parallel long immediate
        splk  #z+FILT_ORDER, cbsr1
        splk  #0bh, cbcr
        splk  #FRAME-1, brcr
        lar   ar5, #x                  ; lar = load auxiliary register
        lar   ar4, #y
        lar   ar3, #z
        rptb  loopend-1
        lacc  *+, 9, ar3       ; ar5=&x[i]          lacc: load acc with shift
        sach  *-, 7, ar3       ; ar3=&z[0]=>&z[FILT_ORDER]
        zpr                    ; zpr = zero product register
        rpt    #FILT_ORDER-1
        macd  #coef, *-        ;ar3=&z[FILT_ORDER]
        apac
        dmov  *,ar4
        sach  *+,7,ar5         ;ar4=&y[i]
loopend ...
```

Coefficients update in LMS adaptive filtering

- $a_k[i+1] = a_k[i] + \underbrace{2\,\beta\,e[i]}_{\text{precomputed}} * x[i-k]$

  $y[i] = \text{Sum}(k=0, N-1)\ a_k[i] * x[i-k]$

- y[i] computation is the same to the FIR filtering
- Coefficients update (a_k) needs 1 load, 1 multiply and add, 1 store -> 3 cycles/tap

- **Filter coefficients update assembly routine**
- ***AR2: AK pointer**
- ***AR#: x[i-k] pointer**

```
        LT              err
        MPY             beta
        PAC
        SACH            ERRF, 1             ; ERRF = BETA*E[I]
        MPY     *-, AR2             ; P = ERRF * X[I-N+1]
        RPTB            LOOP
        ZALR            *-, AR3             ; LOAD AK
        MPYA    *-, AR2             ; P = ERRF * X[I-K-1]
                        ;ACC = AK[I]+ERRF*X[I-K]
LOOP:   SACH    *+      ; STORE AK[I + 1]
        ZALR            *, AR2  ; FINAL UPDATE A0[I]
        APAC
        SACH            *+
```

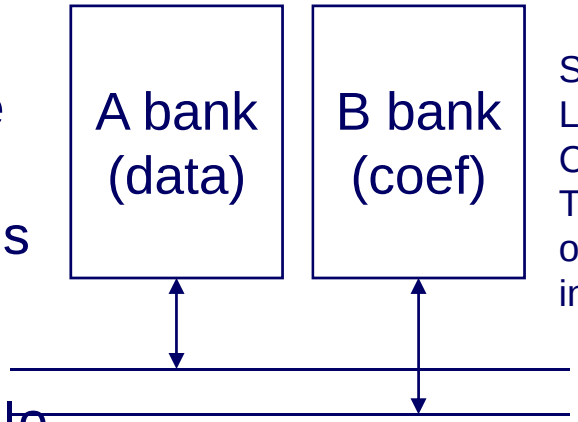
- **Note: *+ means auto-increment, *- represents auto-decrement
- ZALR: zero low accumulator and load high accumulator

4. Programmable DSP vs RISC CPU

- **Nowadays there are many powerful general purpose CPU's**
  - Superscalar RISC with MMX
- **But programmable DSP's are more efficient because**
  - The functional units and instructions are dedicated to signal processing. This means that it is not good for text manipulation and so on.
  - The instructions are application specific and densely encoded (low power)
- **But programmable DSP's are less flexible, and compiler performance may not be good.**
  - RISC instructions are more general and flexible, so compiler performance is better
  - Superscalar architecture achieves the speed up fairly independent of the applications

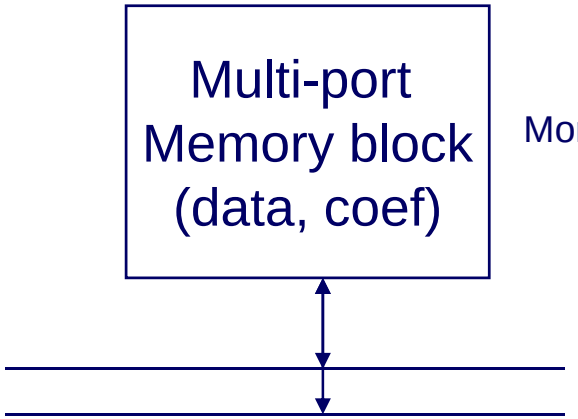
Difference between superscalar RISC and DSP

- **Superscalar RISC**
  - Multi-port memory block and one general purpose register file
  - Multiple instruction issues (more flexible)
  - Intended for general purpose speed up (flexible instructions)
  - Intended for general purpose register based operations
  - Much more expensive in terms of HW and power consumption
- **Multiple memory blocks vs multiport memory**



Simple HW
Low power
Complex to use
Two data accesses only when they are in different banks

Programmable DSP



More convenient to use
Complex HW
Larger power consumption

RISC CPU

Performance of DSP over RISC CPU

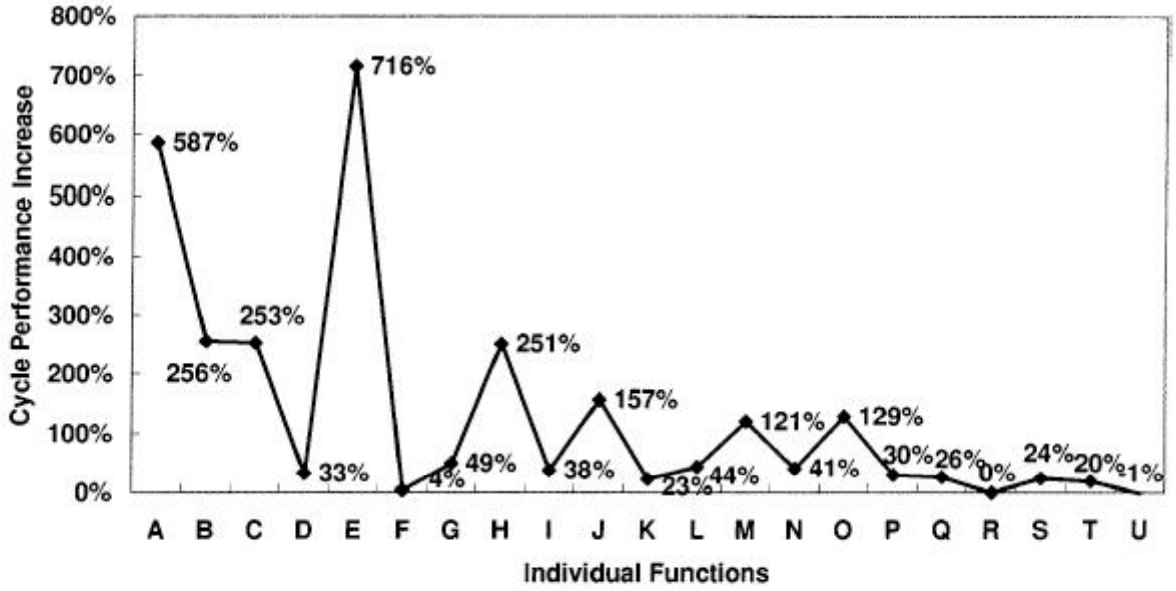


Figure 12. Cycle performance increase of individual functions in QCELP.

Programmable DSP based program development – why difficult?

- **Parallel and pipelined execution of multiple functional units**
  - multiplier-adder-shifter, address generation units, ..
- **Bank control and distributed and special registers**
  - Need to choose which memory block to store
    - Two simultaneously used ones should be stored at different blocks
  - Several special purpose registers for dedicated use (very inconvenient for compilers)
- **Scaling**
  - For fixed-point implementation
- **Limited memory space, HW stack, non-orthogonal instruction encoding (dense instruction encoding)**

Note: Compiler Model

Single general purpose register file, Single function for each instruction

Current and future DSP

- **Application specific DSP's**
  - C54x, C55x – special instructions for Viterbi algorithm, FFT, LMS adaptive filter, linear-phase FIR filtering (2tap/sample)
  - Much higher performance for wireless applications (Viterbi decoding, CELP vocoder)
- **High performance VLIW DSP**
  - C6000 series
  - Good C language support
  - Base station for wireless communication
- **GPP + DSP SoC for communication terminal**
  - TI OMAP: ARM9+C55x/C54x

TI OMAP: Wireless phone organization