Advanced Digital System Design

Dr. Edward Gatt

Introduction to Real Time DSP

- Signals can be of 3 types – continuous-time analogue signals, discrete-time signals & digital signals
- Digital signal processing is concerned with the digital representation of signals and the use of digital hardware to analyse, modify and extract information from this signal
- Advance in digital technology allows real-time implementation of sophisticated DSP algorithms – DSP is not only replacing analogue methods but also analogue techniques which are either difficult or impossible

Advantages of DSP over analogue

- **Flexibility**: - Functions of DSP system can easily be modified or upgraded via software – eg. digital camera using JPEG can be reprogrammed to use higher quality JPEG2000

- **Reproducibility**: - Performance of a DSP system can be repeated precisely from 1 unit to another while analog components suffer from component tolerances. Also, digital signals can be reproduced without loss of quality

- **Reliability**: - Memory and Logic of DSP does not deteriorate with age. There is no drift as found in analogue systems

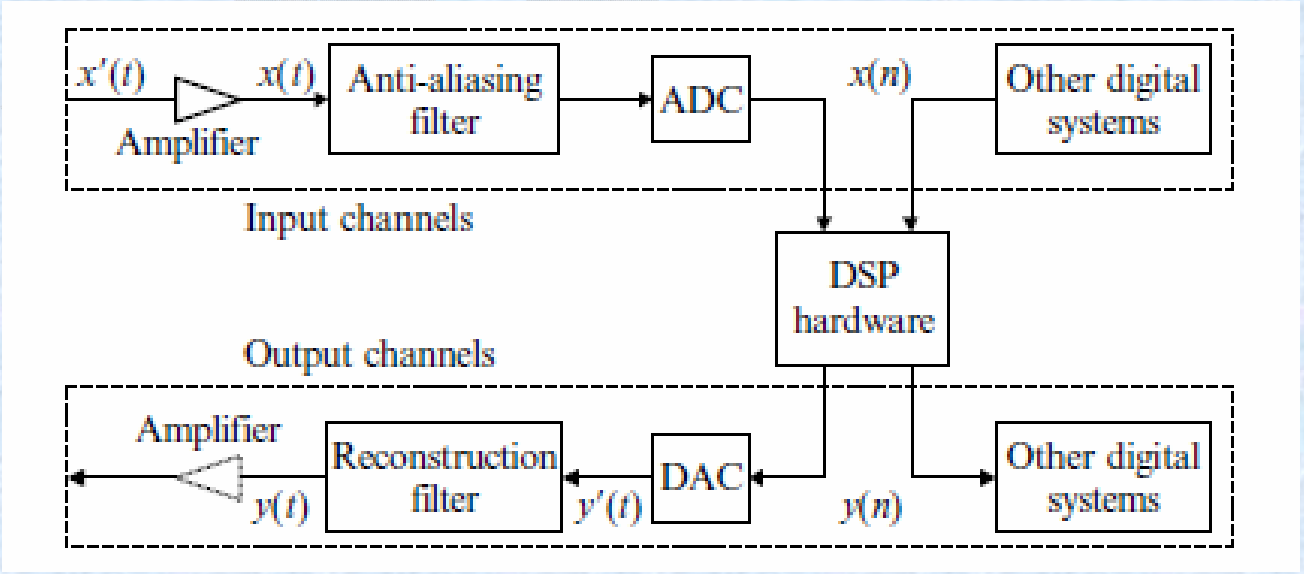
- **Complexity**: - Using DSP, complex applications such as speech recognition or image processing is possible for lightweight and low power portable devices. Also they allow functions as error coding, data transmission and storage, data compression and other functions impossible for analogue

DSP Limitations

- Bandwidth of a DSP system is limited by the sampling rate and hardware peripherals
- Initial design cost of a DSP system may be expensive, especially for large bandwidth signals
- DSP algorithms are implemented using a fixed number of bits, which results in a limited dynamic range and produces quantization and arithmic errors

Elements of a DSP System

Analogue signal is converted to a Digital signal.
Data is processed by the DSP hardware in digital form.
Data may need to be converted back into Analogue form.
For some real-time applications, data may already be in digital form and the data may not need to be output in Analogue form.

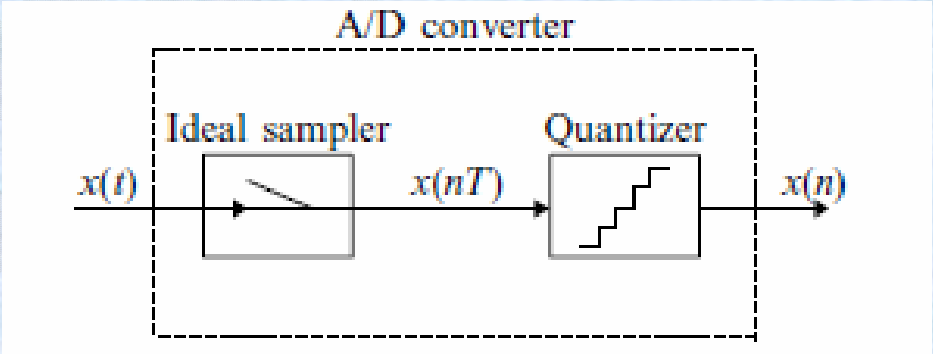


Signal Conditioning

- Analogue signal are picked up by sensors that convert properties to electrical signals
- Sensor output normally needs amplification
- Gain is determined such that the signal matches the dynamic range of the ADC
- In practice, it is difficult to set up an appropriate fixed gain because the level of the signal maybe unknown and changing with time
- Automatic Gain Control (AGC) with time-varying gain determined by DSP hardware can be used to effectively solve this problem

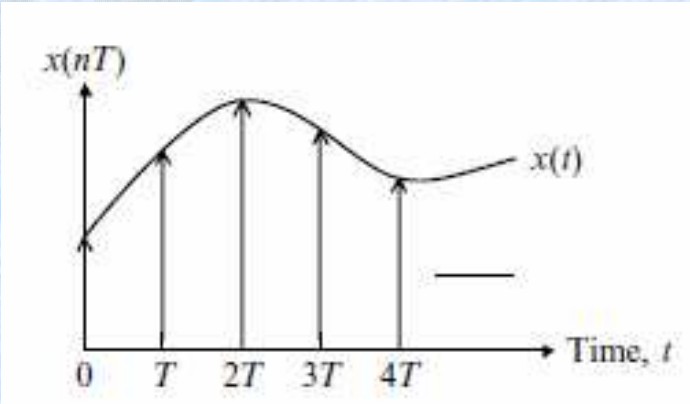
A/D Conversion

- A/D conversion consists of sampling and quantization processes
- Sampling process depicts a continuously varying analogue signal as a sequence of values – done with sample & hold circuit, which maintains sampled level until the next sample is taken
- Quantization process approximates waveform by assigning an actual number (represented by a number of bits) for each sample



Sampling

- Ideal sampler is a switch that is periodically opened and closed every T secs
- The signal is an impulse train with values equal to the amplitude of the analogue signal
- For accurate representation two conditions need to be met -> analogue signal must be band-limited -> sampling freq must be at least twice the max freq in the analogue signal (Shannon's Sampling Theorem)

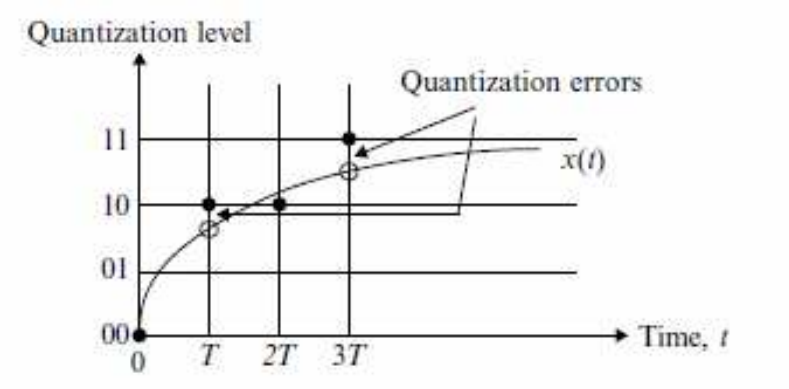


Sampling [2]

- Min $f_s = 2f_M$ (Nyquist Rate)
- When an analogue signal is sampled at sampling freq f_s, freq components higher than $f_s/2$ fold back in the freq range [0, $f_s/2$]-> aliasing
- In this case, the original analogue signal cannot be recovered
- In most applications, signals contain noise and are not band-limited as noise spreads over a wide band
- e.g. for voice signals -> f_s typically 8 kHz, but freq component in speech is typically higher than 4 kHz -> anti-aliasing filter is used to band-limit the signal
- Anti-aliasing filters are not ideal filters and do not remove all freq components outside the Nyquist interval -> thus we still have aliasing distortion -> also phase response of the filter might not be linear producing signal shifts
- To accommodate practical specifications for anti-alias filters oversampling is performed

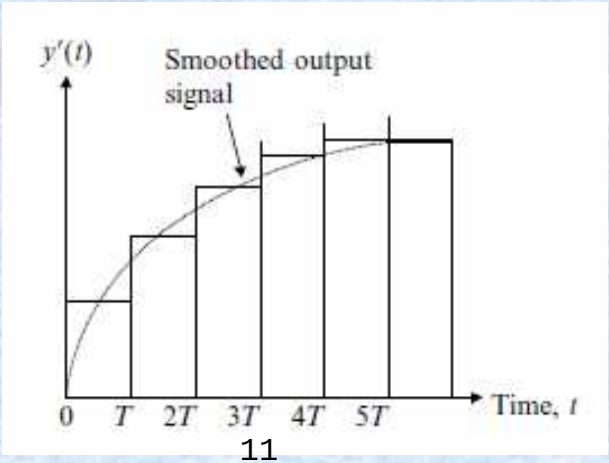
Quantization

- Obvious constraint of physical realisable digital system is that sample values are represented by a finite number of bits
- Discrete-time signal is quantized to a digital signal with a finite number of bits
- e.g. ADC of 8 bits allows 2^8 different levels to represent a sample
- Quantization is the process that represents an analogue-value sample to the nearest level that corresponds to the digital signal
- Quantization produces errors that cannot be removed
- Difference between quantized number and original value is quantization error, which appears as noise in the output



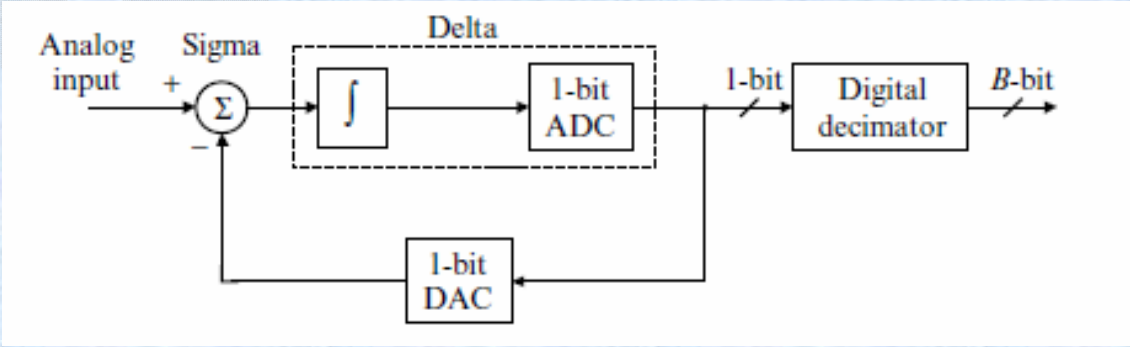
D/A Conversion

- Most commercial DACs are zero-order-hold, -> convert binary input to analogue level and hold value for T secs until next sampling instant
- DAC produces a staircase shape analogue waveform
- Smoothing filter used to smooth staircase-like output signal -> filter maybe same as anti-aliasing filter
- Output of DAC contains unwanted high freq or image components centred at multiples of the sampling freq
- Again ideal filter should have a flat magnitude response and linear phase in the passband



Input/Output Devices

- Two basic ways of connecting A/D and D/A converters to DSP -> serial or parallel
- Serial converters are preferred in practice as parallel converters need to share address and data buses with other different types of devices and with different memory devices running at different speeds hanging on the DSP data bus -> driving the bus might cause problems
- Many applications adopt single-chip devices integrating filters, ADC, DAC.
- Devices sometimes use logarithmic quantizer which must be converted to a linear format for processing -> done via look-up table or simple calculation
- Sigma-delta converters ADC use 1-bit quantizers with a very high sampling rate thus relaxing requirements for anti-aliasing filter -> advantage is high resolution with good noise characteristics at competitive price



DSP Hardware

- DSP require intensive arithmetic operations esp. multiplication and addition
- Options
  - General-purpose microprocessors and microcontrollers
  - General-purpose DSP chips
  - Digital Building Blocks
  - Application Specific Integrated Circuits (ASIC)

	ASIC	DBB	μP	DSP chips
Chip count	1	> 1	1	1
Flexibility	none	limited	programmable	programmable
Design time	long	medium	short	short
Power consumption	low	medium–high	medium	low–medium
Processing speed	high	high	low–medium	medium–high
Reliability	high	low–medium	high	high
Development cost	high	medium	low	low
Production cost	low	high	low–medium	low–medium

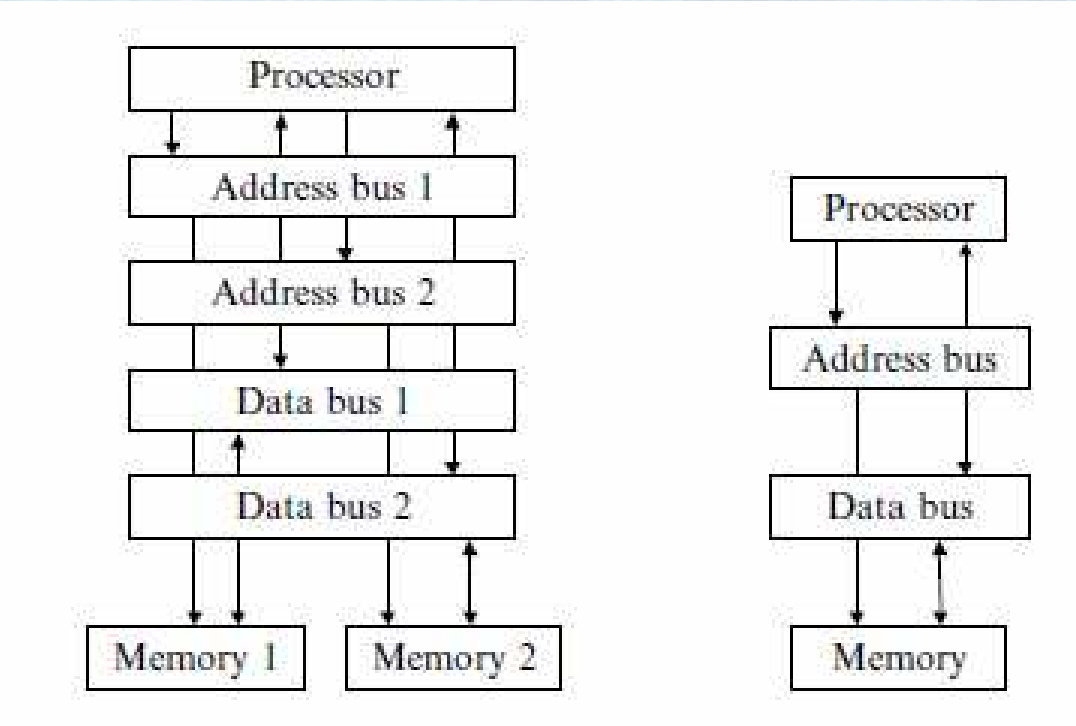
Comparison

- ASIC perform specific tasks and are capable of performing limited tasks much-faster than general-purpose DSP chips because of their dedicated architecture
- ASIC lack programmability to modify the algorithm and are suitable for implementing well-defined and well-tested algorithms
- Problem is that prototyping costs are high, require longer design cycle and lack of re-programmability flexibility

Comparison [2]

- Digital Building Blocks using multipliers, ALUs, sequencers etc.. can perform better than general purpose DSP, but incur higher costs and might be complex to integrate
- General architectures for computers and microprocessors fall into TWO categories
  - Harvard (separate memory for program and data – accessed simultaneously)
  - Von Neumann (assumes one memory for instructions and data)

Harvard vs von Neumann



Von Neumann Architecture

- Most general-purpose microprocessors use von Neumann architecture
- Operations such as add, move, subtract easy to implement but complex instructions multiplication and division are slow as require sequence of shift, addition or subtraction
- Possible for small systems but their performance is weak compared to DSP for complex tasks

DSP Chips

- DSP chip is a microprocessor, whose architecture is optimised for high rate processing
- Chips have been launched by Texas Instruments, Motorola, Lucent Technologies etc.
- Provide flexibility, low-cost and supported by software development tools such as compilers, assemblers, optimizers, linkers, debuggers, simulators and emulators

Fixed- and Floating Point Devices

- Basic distinction between DSP chips is their fixed-point or floating-point architectures
- Fixed-point processors are normally 16-bit or 24-bit while floating-point processors are usually 32-bit
- Typical 16-bit fixed-point processor (TMS320C55x)
- Although coefficients and signals are stored with 16-bit precision – intermediate values (products) maybe kept at 32-bit precision to reduce cumulative round-up errors
- Fixed-point processors are cheaper and faster

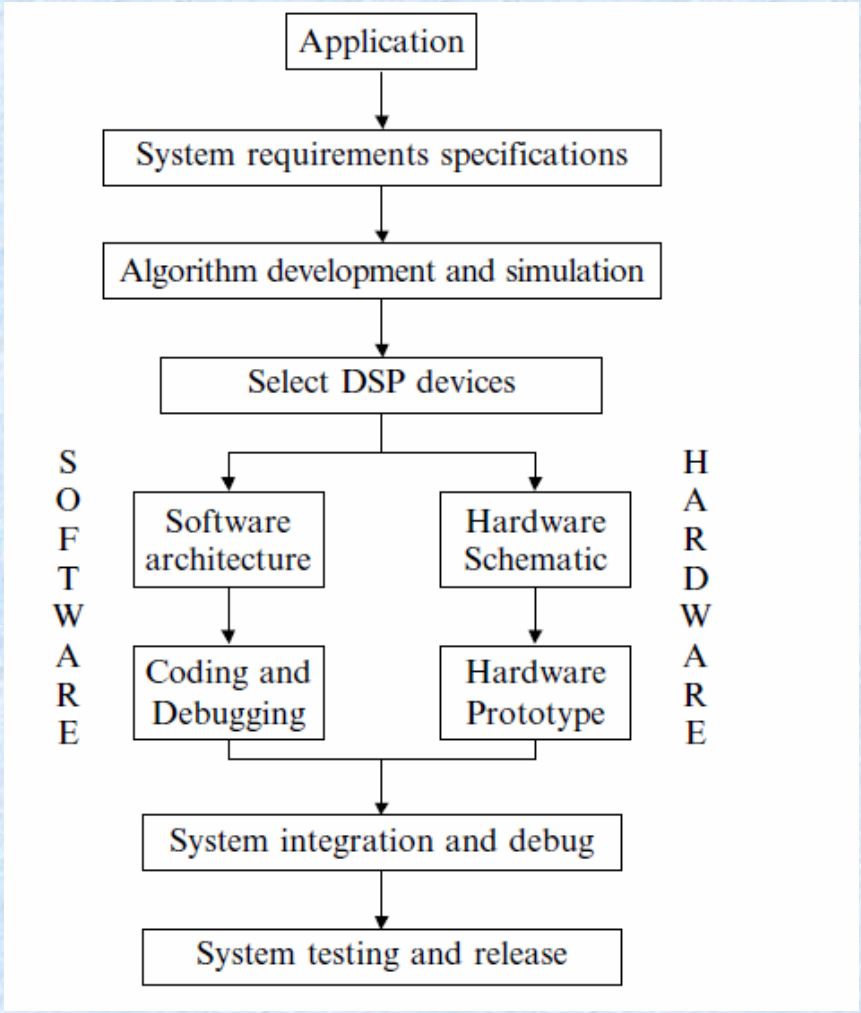
Fixed- and Floating Point Devices [2]

- 32-bit floating point processor (TMS320C3x) stores 24-bit mantissa and 8-bit exponent
- 32-bit gives large dynamic range
- Resolution is still 24-bit
- Dynamic range limitations maybe ignored in design with floating-point DSP while with fixed-point designs, the designer has to apply scaling factors to prevent arithmetic overflow, which is difficult and time-consuming
- Floating-point devices are needed in applications where coefficients vary in time and require large dynamic range or where large memory structures are required
- Their use can be justified where development costs are high and production volumes are low and faster development cycle can outweigh extra cost of DSP itself
- Floating-point DSP chips allow efficient use of high-level compilers and reduce the need to identify system's dynamic range

Real-Time Constraints

- Limitation for Real-Time application is the bandwidth of the system limited by sampling rate
- The processing speed determines the rate at which the analogue signal can be sampled
- For Real-Time operation the signal processing time must be less than the sampling period in order to complete processing before the next sample comes in
- Generally for Real-Time operation bandwidth can be increased by using faster DSP chips, use simplified algorithms, optimize programs and parallelize using multiple chips
- Trade-off between cost and system performance needs to be achieved

DSP System Design



Ideally algorithm is first simulated on a PC so that it can be analyzed off-line using simulated data.

Using high-level languages, significant time can be saved on the algorithm. Also, C programs are portable to different DSP hardware platforms.

It is easy to debug and modify programs.

Input/output operations based on disk files are easy to implement and the behaviour is easy to analyze.

Floating-point and fixed-point behaviour can be compared.

DSP Selection

- Objective is choose the device that meets the project's time-scales and provides cost-effectiveness
- Some decisions can be taken early based on computational power, resolution and cost
- Efficient flow of data into and out of the processor is critical
- For high volume, cheapest solution of chip needs to be sought
- For low to medium volume, trade-off between software development cost and cost of the device needs to be achieved
- Processing speed is normally crucial though memory size and I/O interfacing might be important

Important DSP support

- Software development tools such as assemblers, linkers, simulators and C compilers
- Available boards for software development and testing
- Hardware testing tools, such as in-circuit emulators and logic analyzers
- Libraries

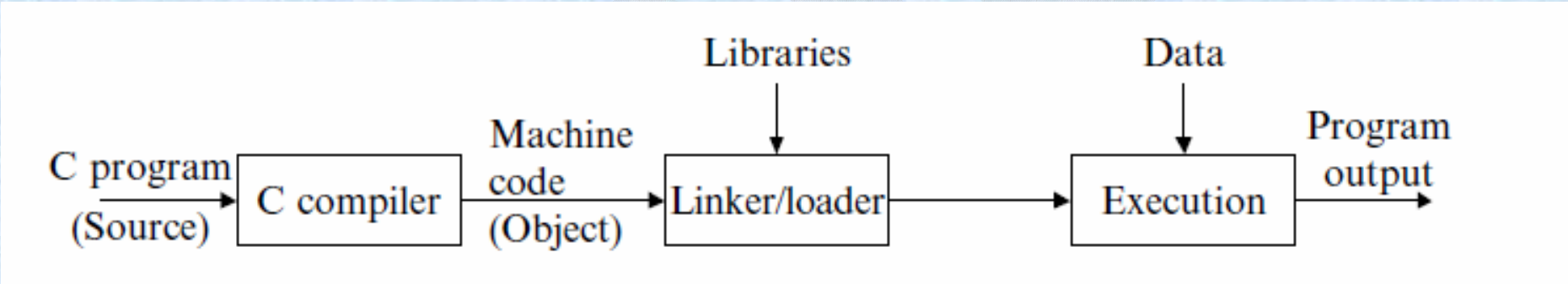
Software Development

- Good software requires to be:
  - Reliable
  - Maintainable
  - Efficient
  - Extendible

Software Development [2]

- Writing and Testing DSP code is highly iterative
- It is important to use modularity for debugging and testing
- Software can be done via assembly providing possibly better efficiency but is laborious and time-consuming for complex systems
- C programming although inefficient in speed and memory usage is easier and portable
- Today C compiler efficiency has been improved
- Ideal solution is possibly having a mixture of C programming and assembly

High-Level Software Development



TMS320C55x Processor

Dr. Edward Gatt

TMS320C55x Processor

- Part of the fixed-point Texas Instrument family
- Design for Low Power Consumption, Optimum Performance and High Code Density
- Its dual multiply-accumulate (MAC) architecture provides twice the cycle efficiency computing vector products
- Its scale-able instruction length significantly improves the code density

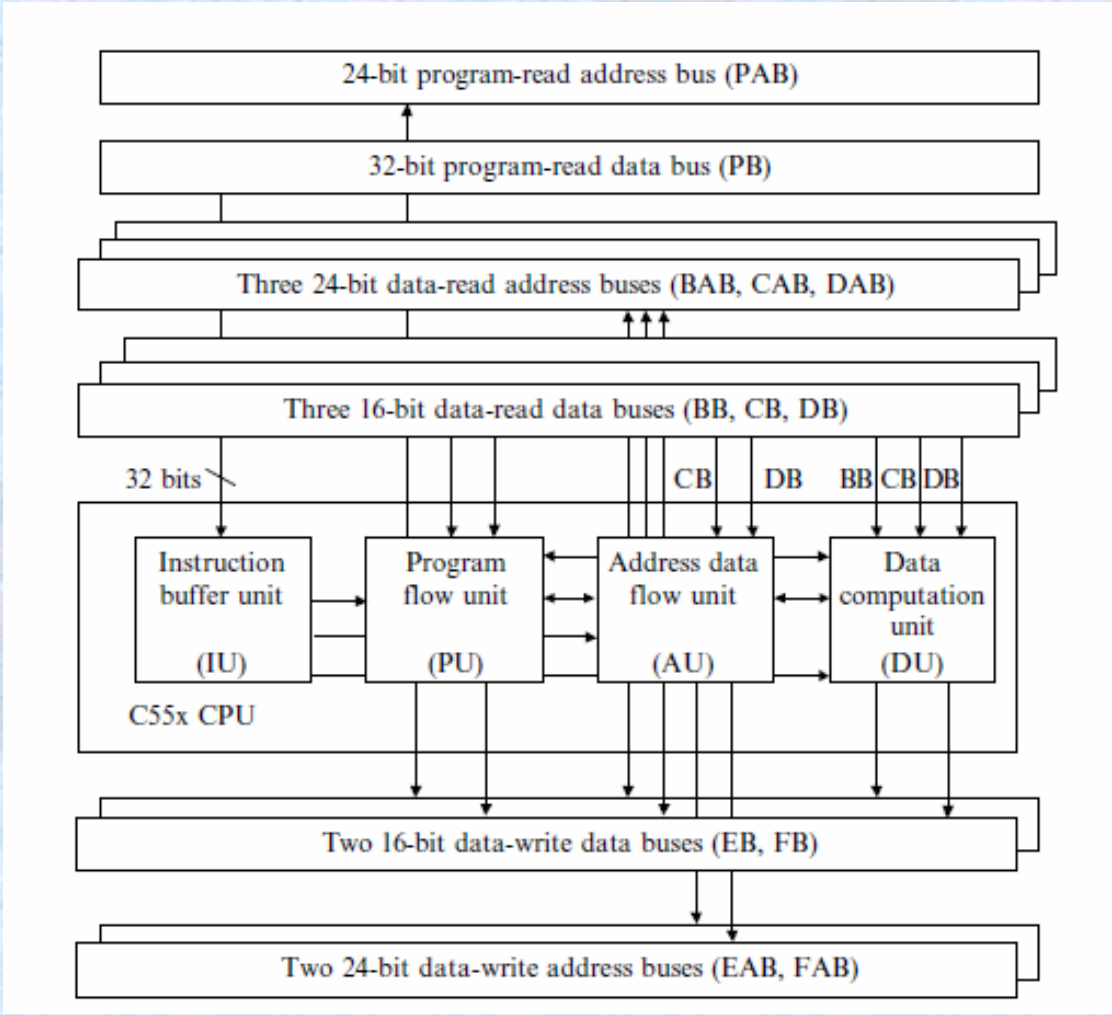
Features

- 64-byte instruction buffer queue that works as a program cache and efficiently implements block repeat operations
- Two 17-bit x 17-bit MAC units can execute dual multiply-and-accumulate operations in 1 cycle
- 40-bit ALU performs high precision arithmetic and logic operations with an additional 16-bit ALU to perform simple operations in parallel with the main ALU
- 40-bit accumulators for storing computational results to reduce memory access
- 8 extended auxiliary registers for data addressing and 4 temporary data registers to ease data processing requirements
- Circular addressing mode support up to 5 circular buffers
- Single-Instruction repeat and block repeat operations of program for supporting zero-overhead looping

Architecture

- C55x CPU includes 4 processing units
  - Instruction buffer unit (IU)
  - Program flow unit (PU)
  - Address-Data flow unit (AU)
  - Data Computation unit (DU)
- Processing Units are connected to 12 different address and data buses

C55x CPU Architecture

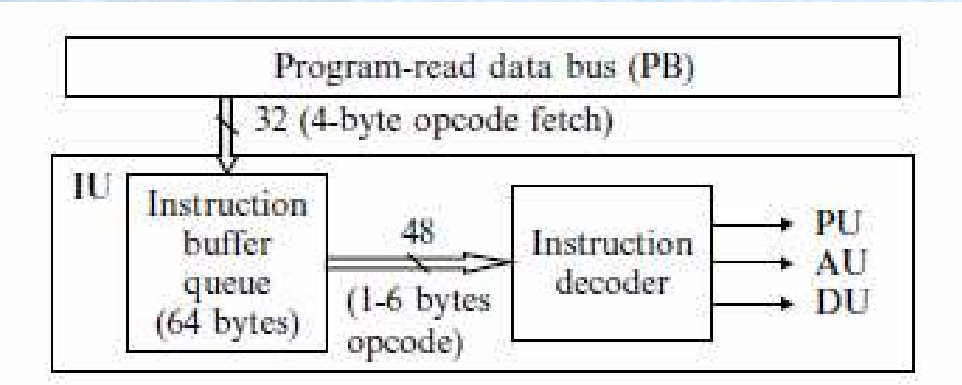


Instruction Buffer Unit (IU)

- The unit fetches instructions from the memory into the CPU
- The instruction set varies in length
- Simple Instructions are coded using 8-bits (1 byte) while more complex ones can contain up to 48-bits (6 bytes)
- In 1-cycle, IU can fetch 4 bytes of program code via 32-bit program-read data bus
- In the same cycle, IU can decode up to 6-bytes of program

Instruction Buffer Unit [2]

- After 4 program bytes are fetched, IU places them in 64-byte instruction buffer
- Simultaneously decoding logic decodes instruction (1-byte to 6-byte long) previously placed in the instruction buffer
- The decoded instruction is passed on to the PU, AU or DU
- IU improves efficiency of program execution by maintaining constant stream of instruction flow between four units within the CPU

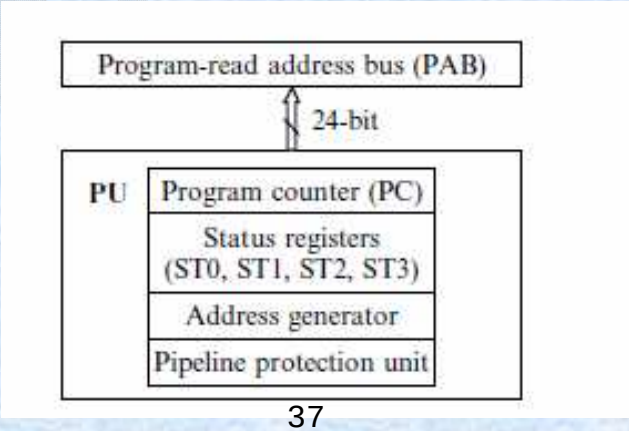


Instruction Buffer Unit [3]

- If IU can hold segment of code within a loop, program execution can be repeated many times without fetching additional code
- This improves loop execution time but saves power consumption by reducing program access from memory
- Another advantage is that IU can hold multiple instructions that are used in conjunction with conditional program flow control
- This minimizes the overhead caused by program flow discontinuities – such as conditional calls and branches

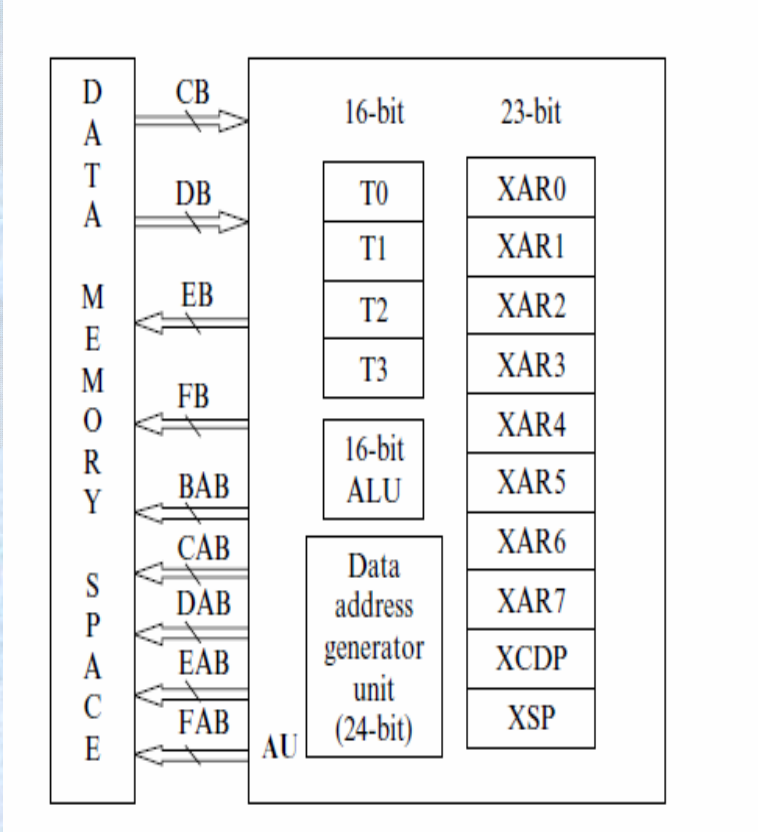
Program Flow Unit (PU)

- PU consists of Program Counter (PC), four status registers, a program address generator and a pipeline protection unit
- PC tracks the C55x program execution every clock cycle
- Program Address Generator produces a 24-bit address that covers 16 Mbytes of program space
- Since most instructions will be executed sequentially, C55x utilizes pipeline structure to improve its execution efficiency
- However instructions like branching, call, return, conditional execution, interrupt will cause non-sequential program address switch
- PU uses a dedicated pipeline protection unit to prevent program flow from pipeline vulnerabilities caused by non-sequential execution



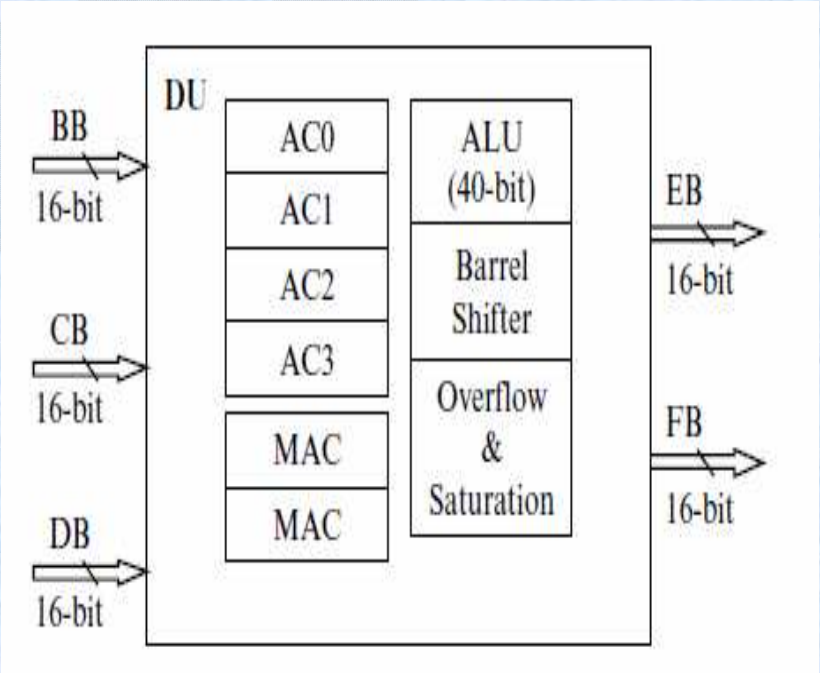
Address-data flow unit (AU)

- AU serves as data access manager for the data read and data write buses
- AU generates the data-space addresses for data read and data write
- AU includes eight 23-bit extended auxiliary registers, four 16-bit temporary registers, 23-bit extended coefficient data pointer, 23-bit extended stack pointer, 16-bit ALU for simple arithmetic operations
- Temporary registers improve compiler efficiency by needing less memory access
- AU allows two address registers and a coefficient pointer to be used together for processing dual-data and one coefficient per clock cycle
- AU supports 5 circular buffers (to be explained later)



Data Computation Unit (DU)

- DU handles processing for most C55x applications
- DU consists of two MAC units, 40-bit ALU, four 40-bit accumulators, barrel shifter, rounding and saturation control logic
- There are 3 data-read data buses that allow 2 data paths and a coefficient path to be connected to dual-MAC units simultaneously
- In 1 cycle, each MAC unit can perform 17-bit multiplication and a 40-bit addition or subtraction operation with a saturation option
- ALU can perform 40-bit arithmetic, logic, rounding and saturation operations using 4 accumulators
- It can also be used to achieve two 16-bit arithmetic operations in both upper and lower portions of an accumulator at the same time
- ALU can accept immediate values from IU as data and communicate with other AU and PU registers
- Barrel shifting may perform data shift 2^{-32} (shift right) to 2^{31} (shift left)



TMS320C55x Buses

- One 32-bit program data bus
- Five 16-bit data buses
- Six 24-bit address buses
- Program buses include a 32-bit program-read data bus (PB) and 24-bit program-read address bus (PAB)
- PAB carries program memory address to read code from program space
- PB transfers four bytes of program code to IU every clock cycle

TMS320C55x Buses [2]

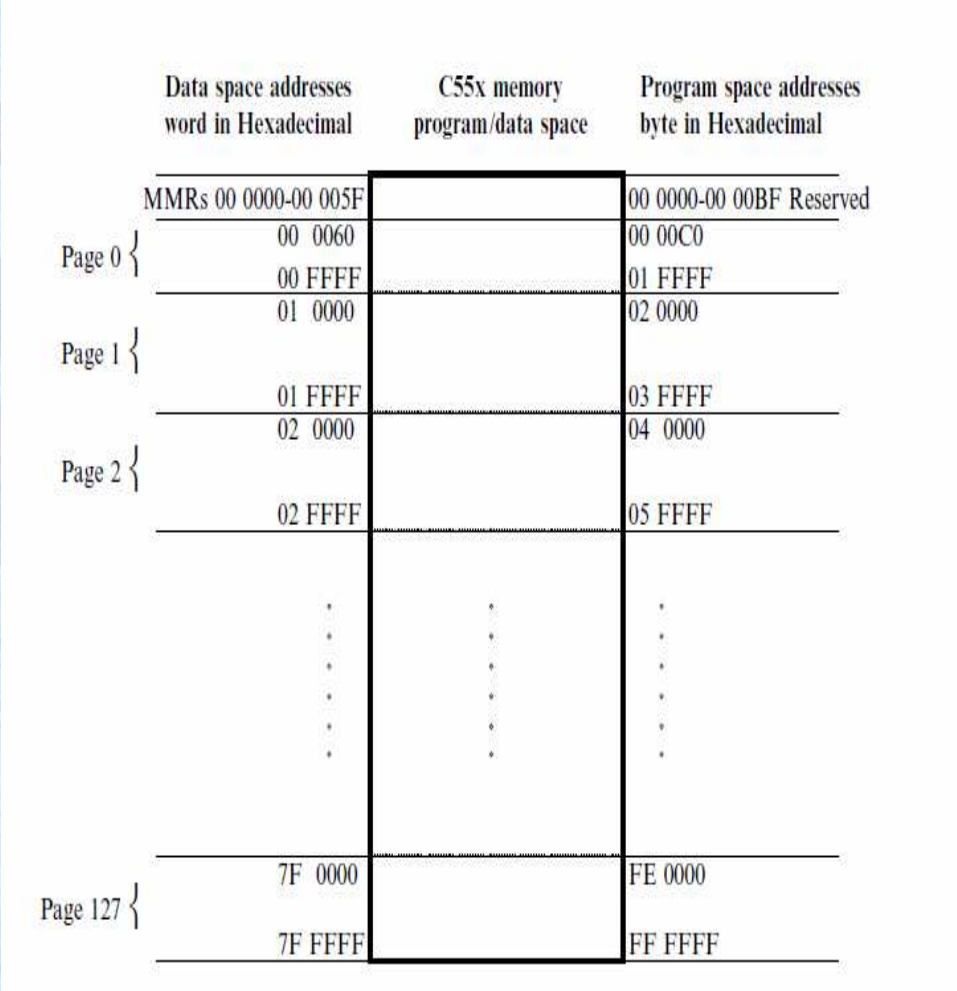
- Data buses consist of three 16-bit data-read data buses (BB,CB,DB) and three 24-bit data-read addresses buses (BAB,CAB,DAB)
- This setup supports 3 simultaneous data reads from data memory or I/O space
- CB and DB can send data to the PU,AU and DU
- BB works only with DU
- Primary function of BB is to connect memory to dual MAC so that some specific operations can access all three data buses, eg fetching 2 data and 1 coefficient

TMS320C55x Buses [3]

- Data write operations are carried out using two 16-bit data-write buses (EB,FB) with corresponding 24-bit address buses (EAB,FAB)
- For 16-bit write only EB is used and both are used for 32-bit write
- C55x has 12 buses – program bus carries instruction code and immediate operands from program memory, while data buses connect various units
- Architecture maximizes processing power by maintaining separate memory bus structures for full-speed execution

TMS320C55x Memory Map

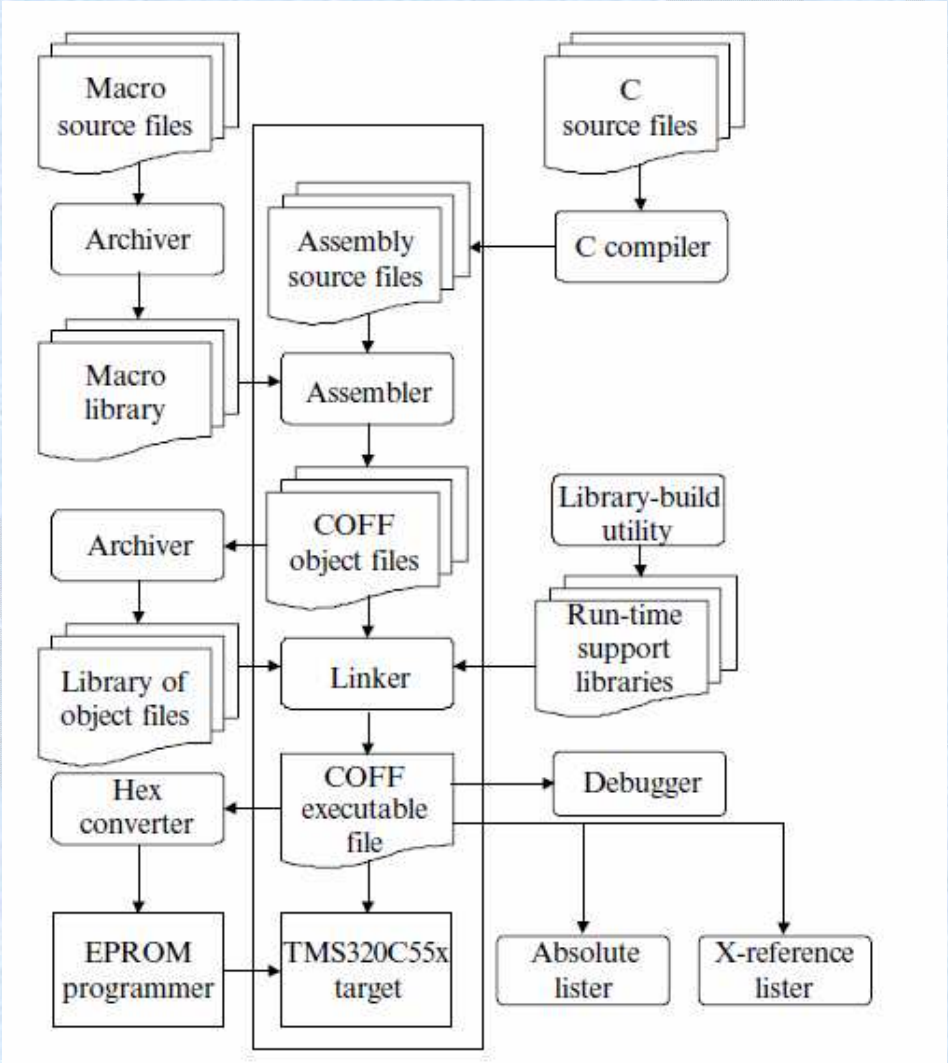
- C55x uses unified program, data and I/O memory configuration
- 16 Mbytes of memory available for program and data
- Program space for instruction codes, data space used for general-purpose storage and I/O space used for duplex communication with peripherals
- Data space divided in 128 data pages of 64 k words
- Memory block 0 -> 0x5F reserved for memory mapped registers



TMS320C55x Software

- Software includes C compiler, assembler, linker, archiver, hex conversion utility
- Debugging tools could be simulator or emulator
- C compiler generates assembly code from C source files
- Assembler translates assembly into machine language object files
- Assembly tools use common object file format to facilitate modular programming (COFF) and allows programmer to define the system's memory map at link time
- This maximizes performance by enabling the programmer to link the code and data objects into specific memory locations
- Archiver allows collection of groups of files into single file
- Linker combines object files and libraries into a single executable COFF object module
- Hex conversion utility converts a COFF object file into a format that can downloaded to an EPROM programmer

Software Development Flow and Tools



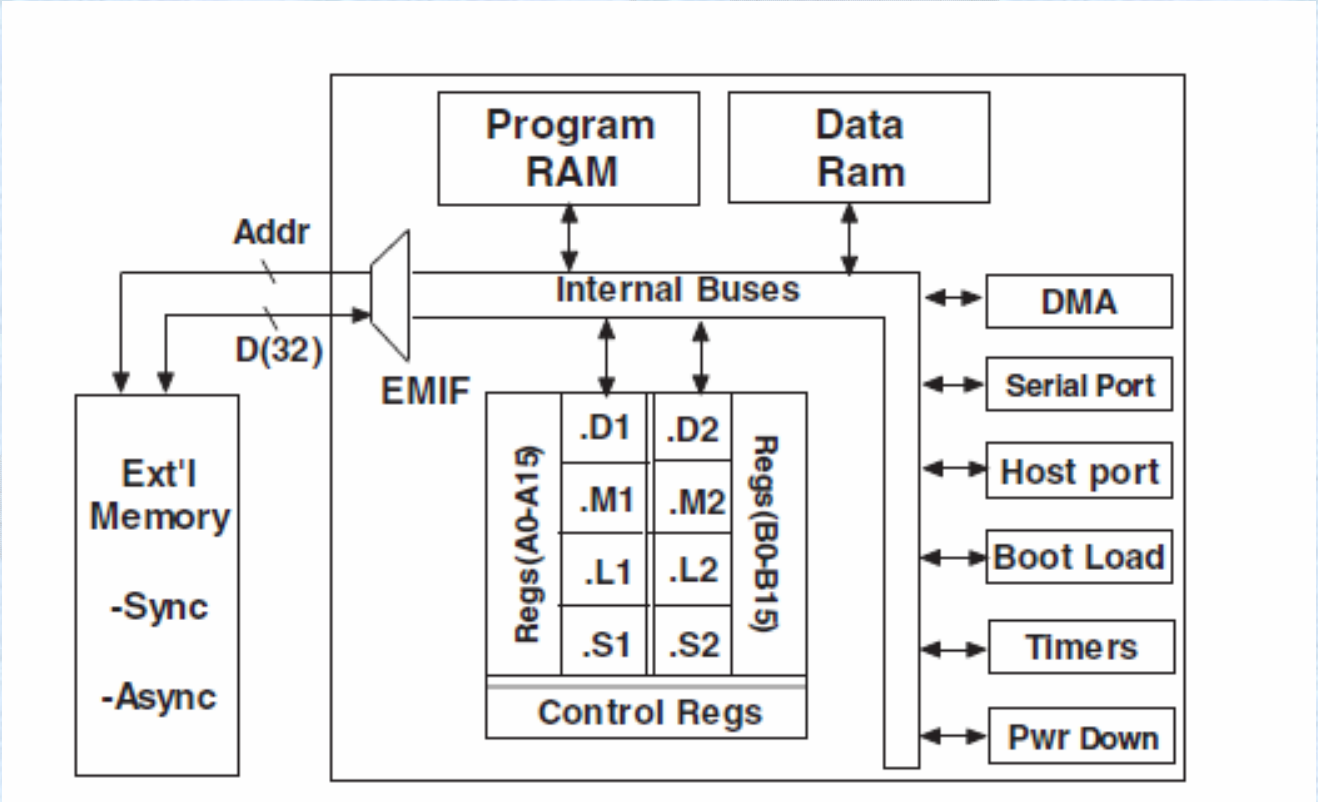
TMS320C6x Architecture

Dr. Edward Gatt

C6x CPU

- The C6x CPU consists of eight functional units divided into 2 sides (A & B)
- Each side has a .M unit (for multiplication operation), a .L unit (for logical & arithmetic operations), a .S unit (used for branch, bit manipulation and arithmetic operations), a .D unit (used for loading, storing and arithmetic operations)
- Instructions such as ADD can be done by more than 1 unit
- There are sixteen 32-bit registers per side
- Interaction with the CPU are done through these registers

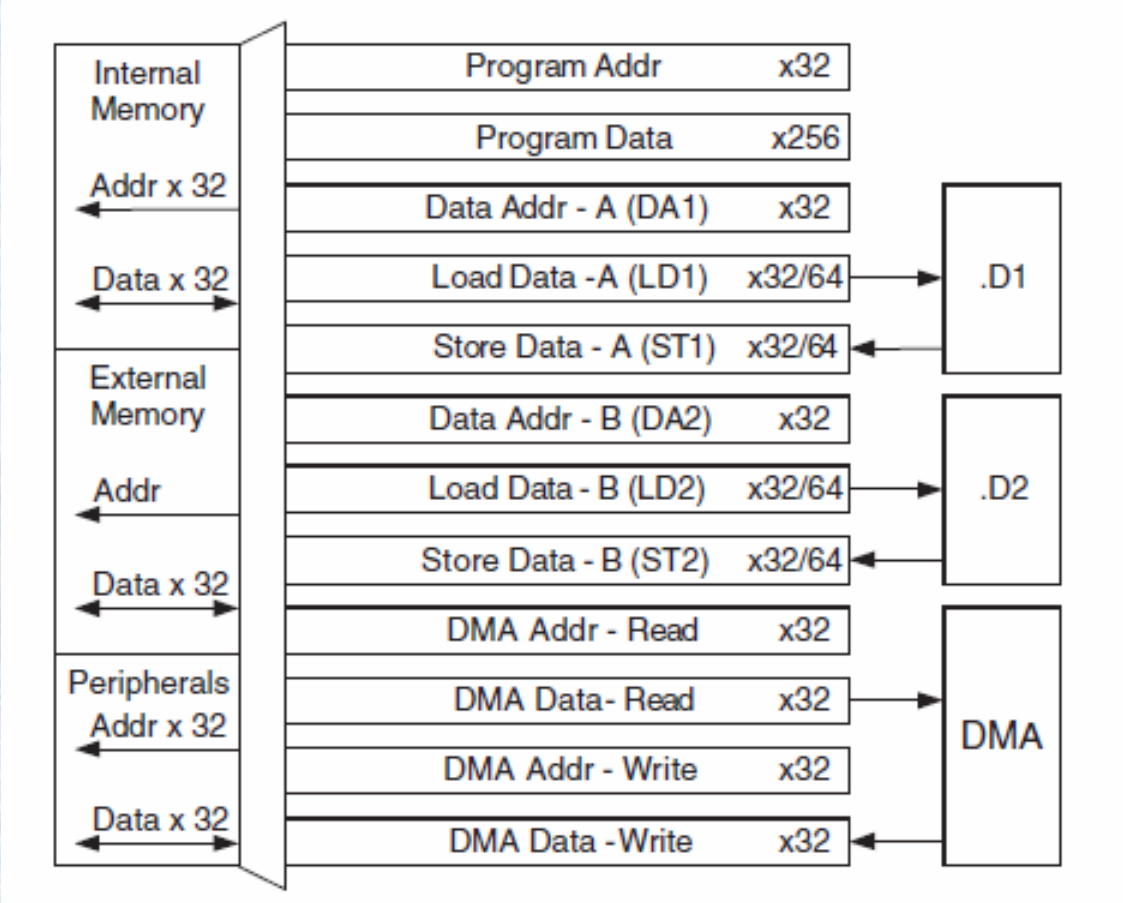
C6x CPU [2]


Program RAM
Data Ram
Addr
D(32)
Internal Buses
EMIF
DMA
Serial Port
Host port
Boot Load
Timers
Pwr Down
Ext'l Memory
-Sync
-Async
Regs(A0-A15)
.D1
.M1
.L1
.S1
.D2
.M2
.L2
.S2
Regs(B0-B15)
Control Regs

Internal Buses

- Internal buses consist of a 32-bit program address bus, a 256-bit program data bus accommodating eight 32-bit instructions
- Two 32-bit data address buses (DA1, DA2), two 32-bit load data buses (LD1, LD2), two 32-bit store data buses (ST1, ST2)
- In addition, there are a 32-bit DMA data bus plus 32-bit DMA address bus

Internal Buses [2]



Peripherals

- External Memory Interface (EMIF) provides necessary timing for accessing external memory
- DMA allows movement of data from one place in memory to another without interfering with the CPU operation
- Boot Loader boots loading of code from off-chip memory or HPI to internal memory
- Multichannel Buffered Serial Port (McBSP) provides high-speed multichannel serial communication link
- Host Port Interface (HPI) allows host to access internal memory
- Timer provides two 32-bit counters
- Power Down unit used to save power for durations when CPU is inactive

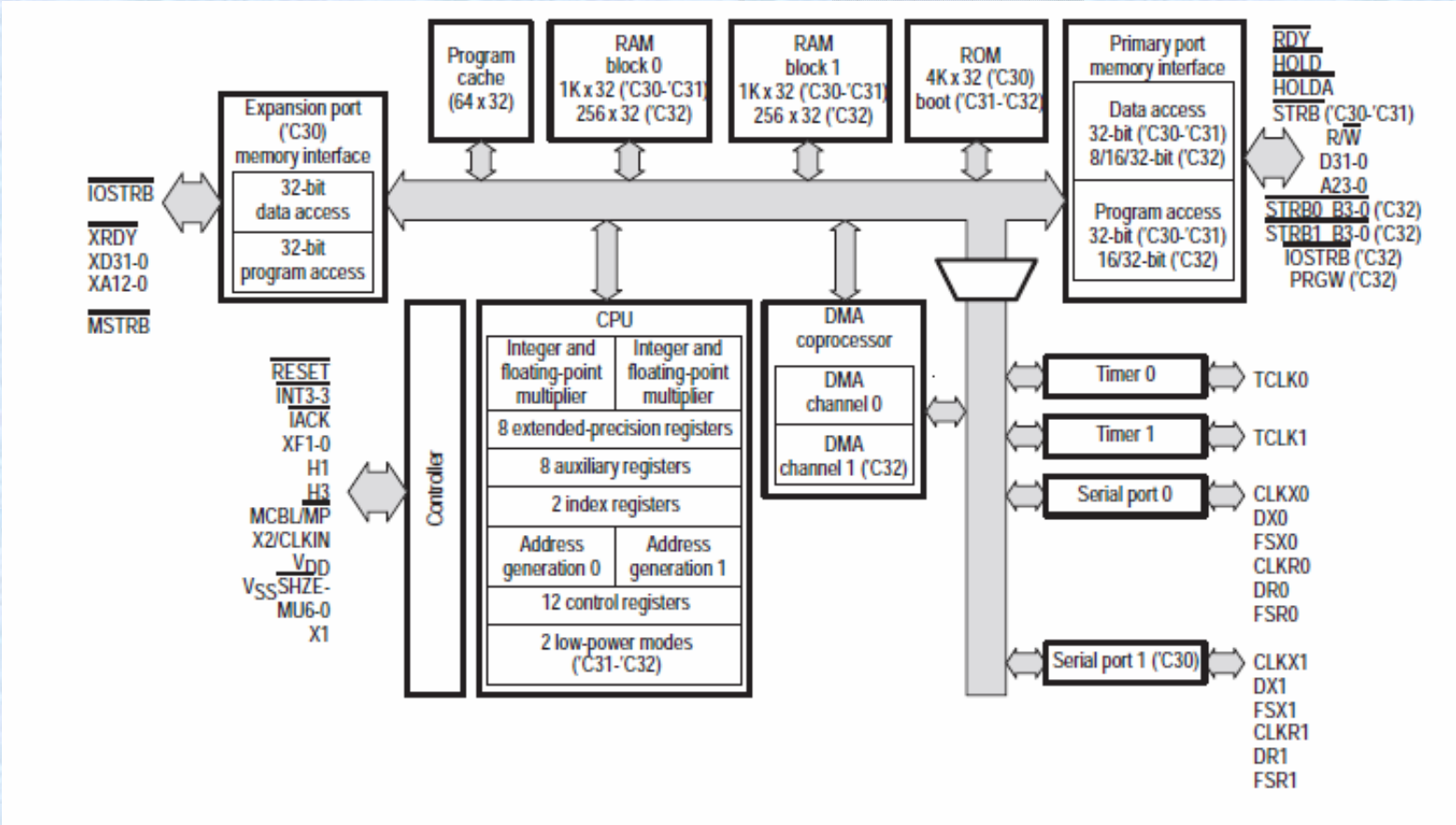
TMS320C3x Architecture

Dr. Edward Gatt

C3x Architecture

- Can perform parallel multiply and arithmetic logic unit (ALU) operations on integer or floating-point data in a single cycle
- Provide high performance via:
  - General-purpose register file
  - Program Cache
  - Dedicated Auxiliary Register Arithmetic Units (ARAU)
  - Internal dual-access memories
  - One direct memory access (DMA) channel supporting concurrent I/O
  - Short machine-cycle time

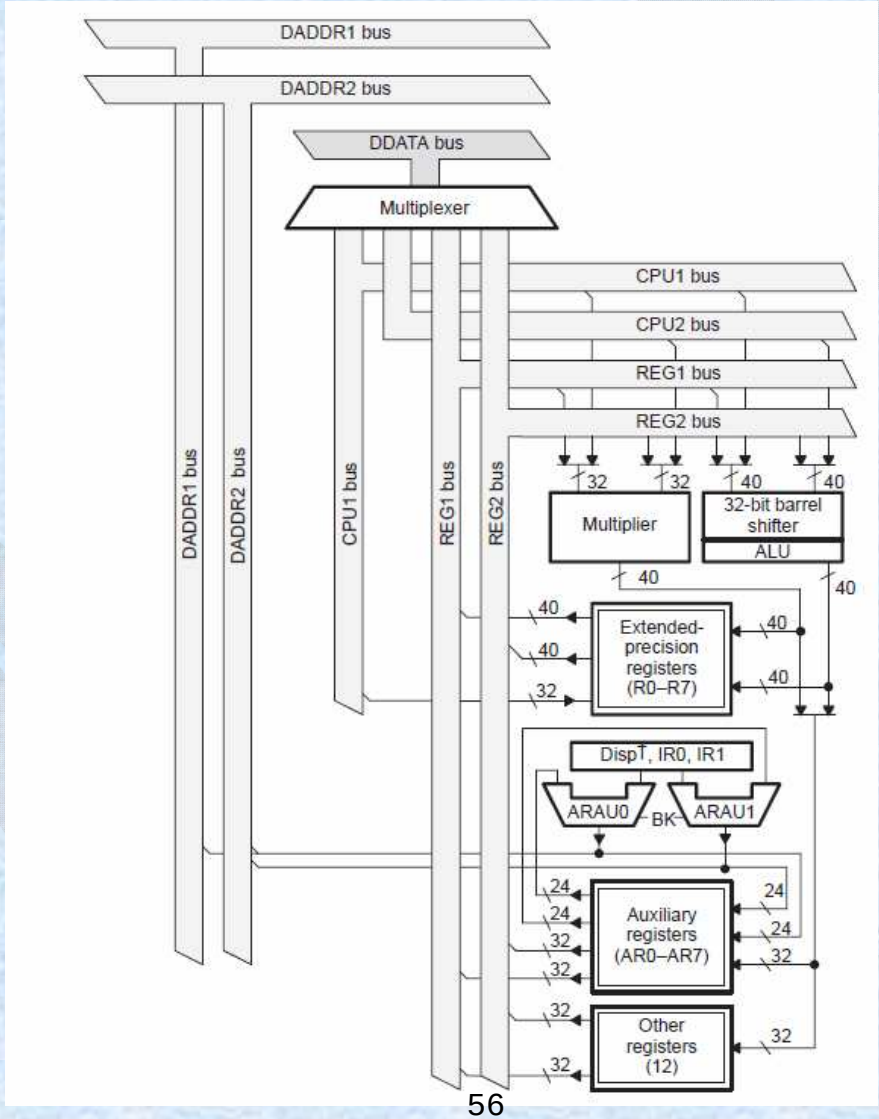
C3x Block Diagram



C3x CPU

- Components include:
  - Floating-point/integer multiplier
  - ALU
  - 32-bit barrel shifter
  - Internal buses (CPU1/CPU2 and REG1/REG2)
  - Auxiliary Register Arithmetic Units (ARAUs)
  - CPU register file

C3x CPU diagram



Floating-Point/Integer Multiplier

- Multiplier performs single-cycle multiplications on 24-bit integer and 32-bit floating point values
- Architecture allows multiplication operations at 33ns instruction cycles
- Higher output can be attained by parallelizing multiply and ALU operation in a single cycle
- For floating point inputs are 32-bit and result is 40-bit floating point
- For integer inputs are 24-bit and result is 32-bit

ALU

- ALU performs single-cycle operations on 32-bit integer, 32-bit logical and 40-bit floating point data, including single cycle integer and floating-point conversions
- Results are 32-bit integer or 40-bit floating-point
- Barrel shifter can shift up to 32-bit left or right in 1-clock
- Four internal buses (CPU1, CPU2, REG1 and REG2) carry two operands from memory and two operands from the register file, allowing parallel multiplies and add/subtracts on 4 integer or floating point operands per cycle

Auxiliary Register Arithmetic Units (ARAUs)

- ARAU0 and ARAU1 can generate two addresses per cycle
- ARAUs operate in parallel with the multiplier and ALU
- Support addressing with displacements, index registers, circular and bit-reversed addressing (covered later on)

CPU Register File

- Extended-Precision Registers can store and support operations on 32-bit integers and 40-bit floating point numbers
- Auxiliary Registers are 32-bit and are accessed by CPU and modified by ARAUs. Their primary function is generation of 24-bit addresses
- Data-page pointer is a 32-bit register used for direct addressing
- Index Registers (32-bit) contain the value used by ARAU to compute indexed addressing

CPU Register File [2]

- System-Stack Pointer (32-bit) contains address of the top of the system stack
- Status register contains global information relating to the sate of the CPU (used for flagging conditions)
- CPU/DMA Interrupt-Enable Register (32-bit)
- CPU Interrupt Flag Register (32-bit)
- I/O flag register controls function of the dedicated external pins
- Repeat-counter (32-bit) specifies the number of times a block code is repeated
- Program-counter (32-bit) contains address of next instruction to fetch – NOT PART OF CPU REGISTER FILE
- Instruction-register (32-bit) holds instruction opcode during the decode phase of the instruction – NOT PART OF CPU REGISTER FILE

Fixed point vs Floating point

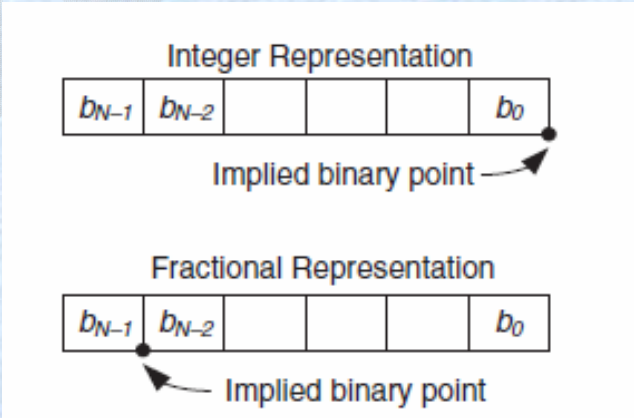
Dr. Edward Gatt

Fixed point vs Floating point

- Fixed point processors – numbers are represented and manipulated in integer form
- Floating point processors – handle both integer arithmetic and floating point - numbers are represented by a mantissa (fractional part) and an exponent part – CPU possesses necessary hardware for manipulating both parts
- Floating point processors are more expensive but slower
- In fixed point processors, dynamic range is a concern as a much narrower range of numbers is represented, which is not a concern for floating point processors
- Fixed point processors usually demand more coding effort than floating point processors

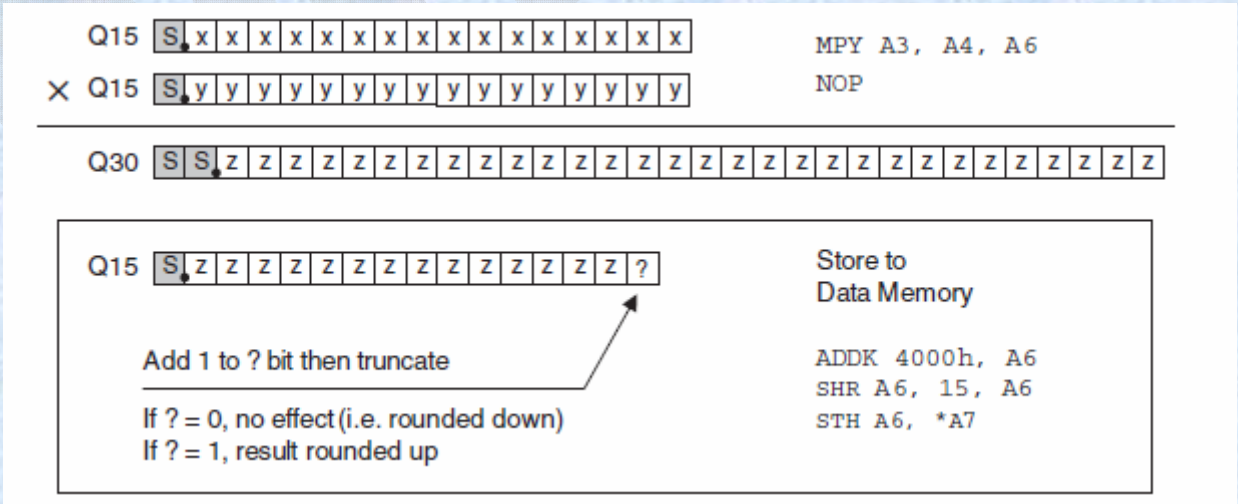
Fixed Point Representation

- 2's Complement allows processor to perform integer addition and subtraction using same hardware
- With unsigned integer representation, sign bit used as extra bit – only positive numbers
- Limitation is dynamic range – eg 16-bit range 32767 -> -32768
- To represent integer with fraction, following number representation is adopted – Q-format

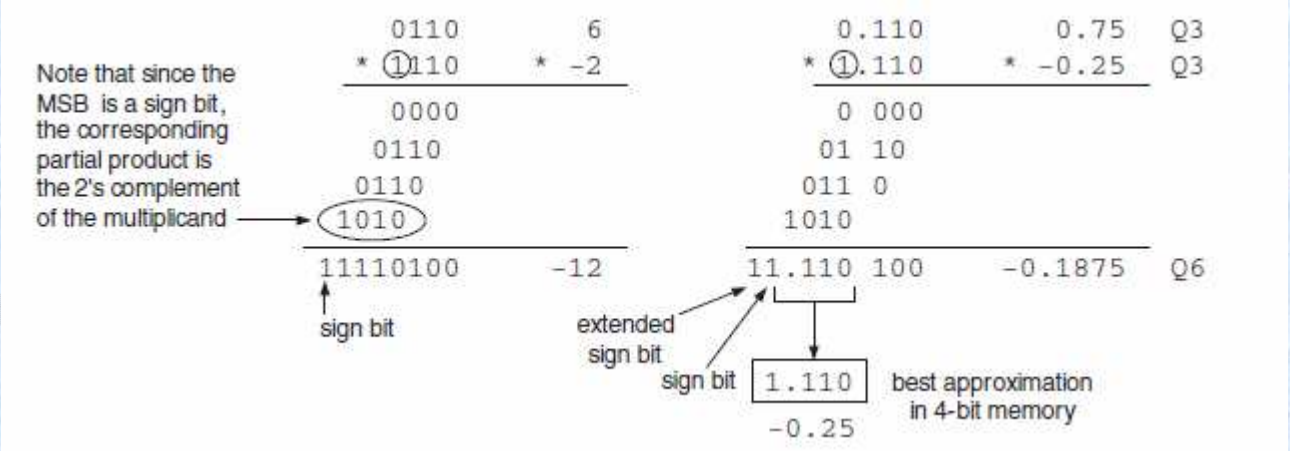


Fixed Point Representation [2]

- Programmer keeps track of implied binary point
- Considering two Q-15 formatted numbers –given a 16-bit bus – numbers have 1 sign-bit plus 15 fractional bits
- Multiplying both numbers, Q-30 formatted number is obtained with bit 31 as sign bit and bit 32 as extended sign bit
- If not all bits can be stored – eg storing answer in 16-bit register, it is best to store MSBs



Fixed Point Representation [3] Multplication

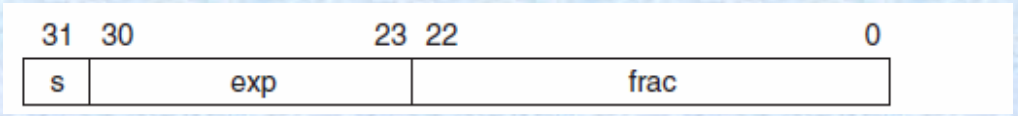


Finite Word Length Effects on Fixed Point DSPs

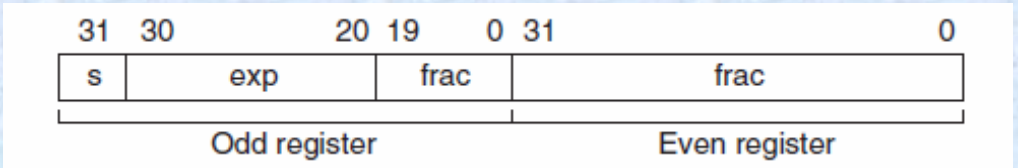
- Due to the fact that memory and registers have finite number of bits, there could be a noticeable error between the desired and actual outcomes on a fixed point DSP
- Effect is similar to input data quantization effect introduced by A/D converter
- Error due to truncation or rounding
- Eg. In FIR or IIR filters, coefficients stored by finite number of bits, positions of poles and zeros are shifted in complex plane

Floating Point Number Representation

- Floating point representation provide a wider dynamic range and there is no need for scaling
- Two types of floating point representation – single precision (SP) and double precision (DP)



- SP 3.4×10^{38} -> 1.175×10^{-38}



- DP 1.7×10^{308} -> 2.2×10^{-308}

Adding Floating Point Numbers

$$a = a_{frac} * 2^{a_{exp}}$$
$$b = b_{frac} * 2^{b_{exp}}$$

$$c = a + b$$
$$= \left(a_{frac} + \left(b_{frac} * 2^{-(a_{exp} - b_{exp})}\right)\right) * 2^{a_{exp}} \quad if \ a_{\exp} \geq b_{\exp}$$
$$= \left(\left(a_{frac} * 2^{-(b_{exp} - a_{exp})}\right) + b_{frac}\right) * 2^{b_{exp}} \quad if \ a_{\exp} < b_{\exp}$$

Although it is possible to compute above computations on a fixed-point processor, it is inefficient as all operations must be done in software

TMS320C55X Detail

Dr. Edward Gatt

Addressing Modes

- Direct Addressing Mode
- Indirect Addressing Mode
- Absolute Addressing Mode
- Memory-Mapped Register Addressing Mode
- Register Bits Addressing Mode
- Circular Addressing Mode

Direct Addressing Mode

- 4 types of direct addressing
  - Data-page pointer (DP) direct
  - Stack-pointer (SP) direct
  - Register-bit direct
  - Peripheral data-page pointer (PDP) direct

Direct Addressing Mode [2]

- DP direct mode
  - Uses 23-bit extended data page pointer (XDP)
  - Upper 7-bits (DPH) determine the data page (0-127)
  - Lower 16-bits (DP) define starting address in data page selected by (DPH)

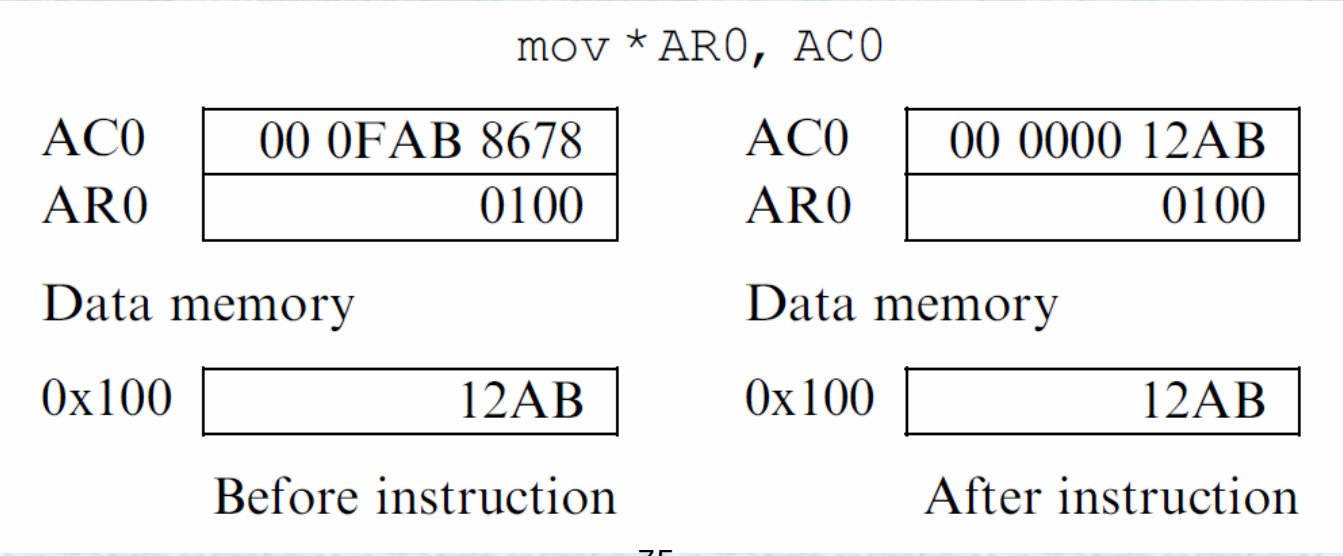
```
mov #k7, DPH   ; Load DPH with a 7-bit constant k7
mov #k16, DP   ; Load DP with a 16-bit constant k16
```

Direct Addressing Mode [3]

- SP direct mode
  - 23-bit address can be formed with the extended stack pointer (XSP) in same way as XDP
  - Upper 7-bits (SPH) defines page
  - Lower 16-bits (SP) determines starting address
- I/O space address mode has only 16-bit address range – 512 peripheral data pages selected by upper 9 bits of PDP – remaining 7-bit determines location inside peripheral page

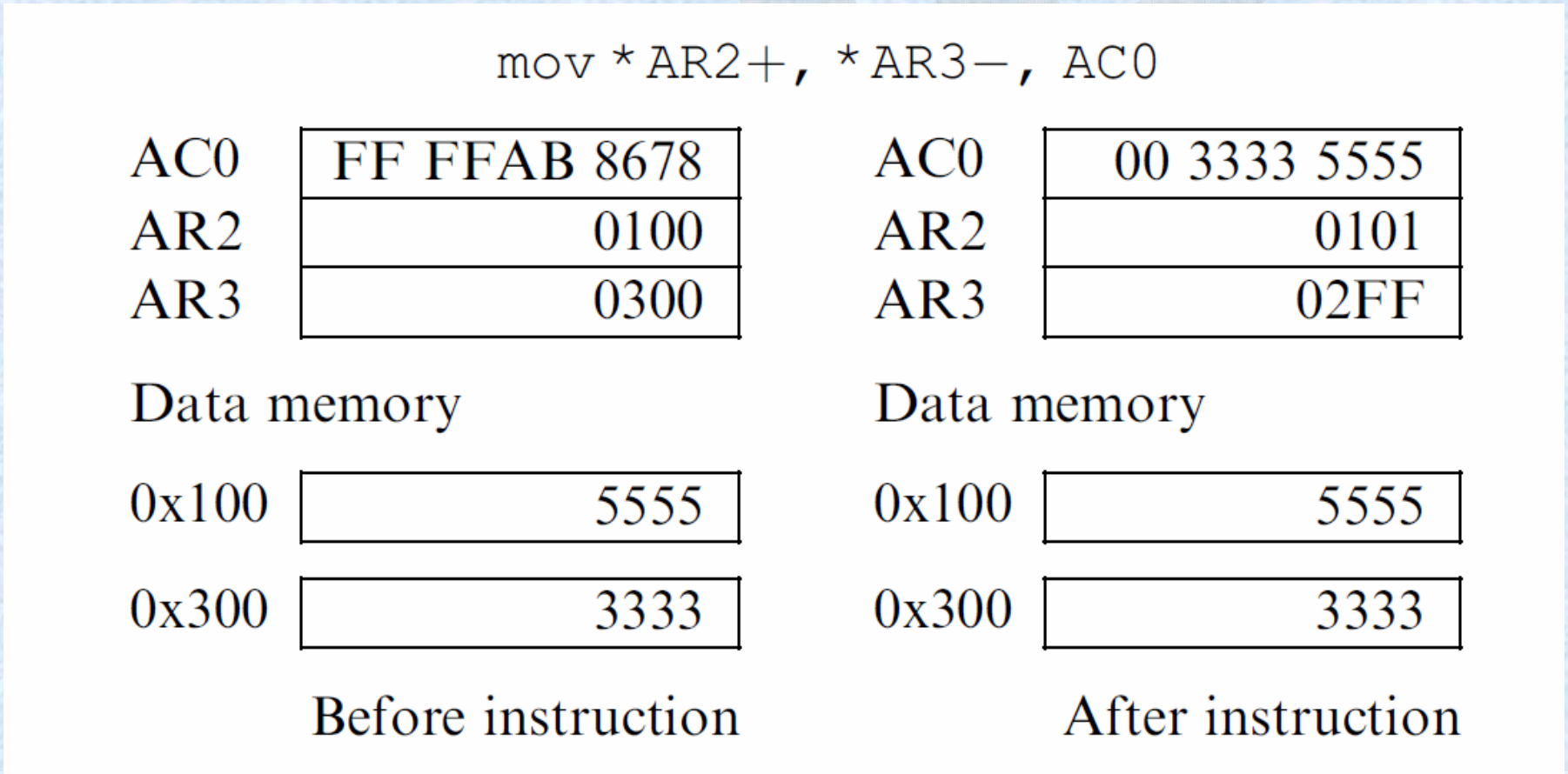
Indirect Addressing Mode

- Indirect Addressing Modes
  - using indexing
  - using displacement
- AR indirect addressing modes uses an auxiliary register (AR0-AR7) to point to the data memory space – upper 7-bits (XAR) points to main page – while lower 16-bits points to data location



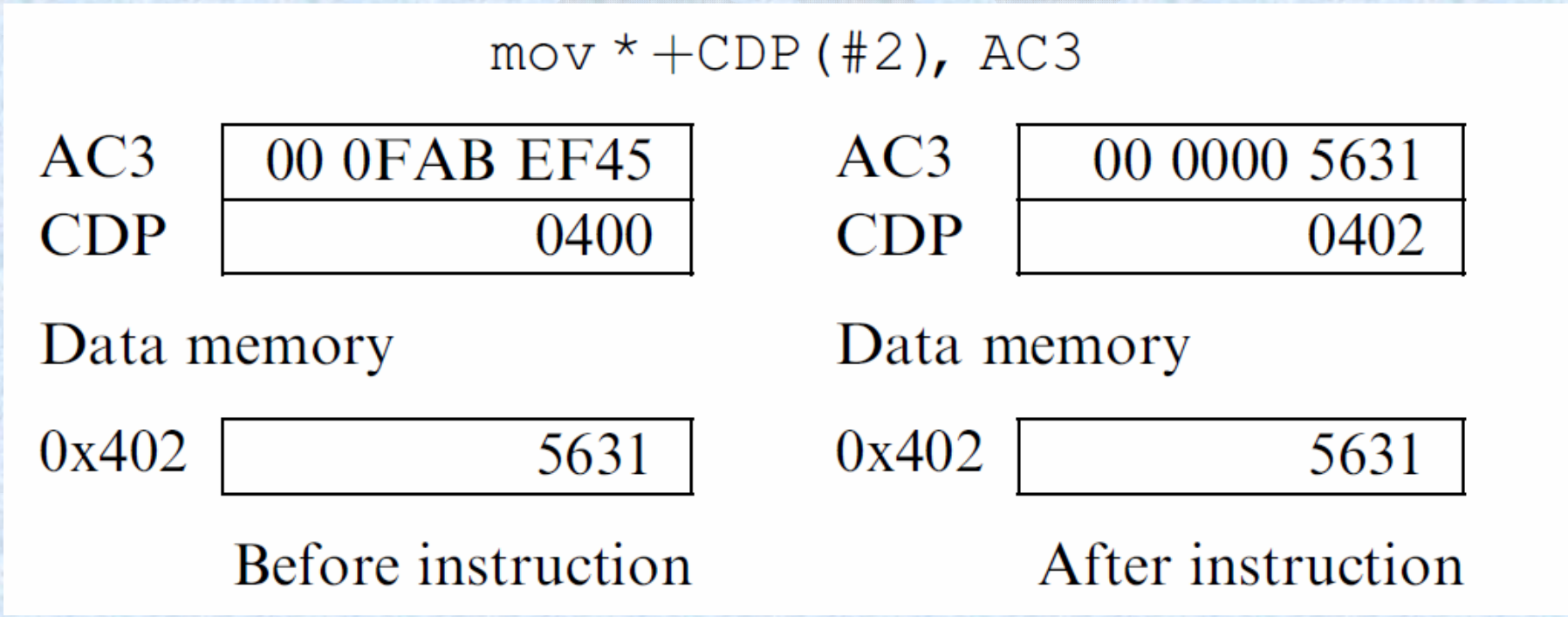
Indirect Addressing Mode [2]

- Dual-AR indirect addressing mode allows two data-memory accesses through auxiliary registers



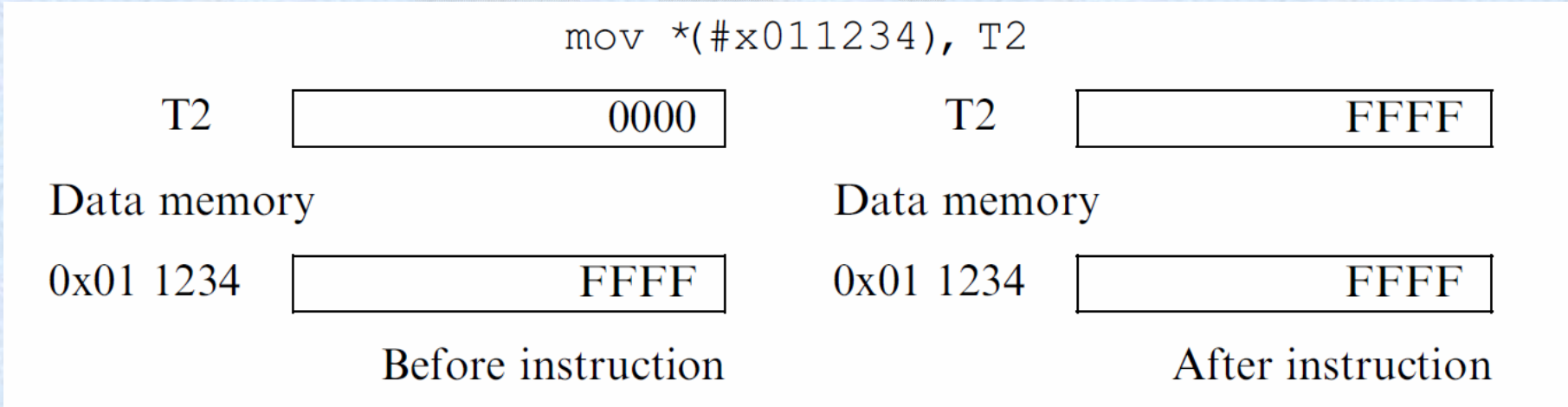
Indirect Addressing Mode [3]

- Extended coefficient data pointer (XCDP) is concatenation of CDPH (upper 7-bits – page address) and CDP (lower 16-bits – data location)



Absolute Addressing Mode

- k16 or k23 absolute addressing modes
- k23 specifies an address as a 23-bit unsigned constant



Memory-Mapped Register Addressing Mode

- Absolute, direct and indirect addressing modes can be used to address MMRs
- MMRs are located in data memory from address 0x0 to 0x5F on page 0

```
mov *abs16(#AR2), T2
```

AR2	1357
T2	0000

Before instruction

AR2	1357
T2	1357

After instruction

Register Bits Addressing Mode

- Both direct and indirect addressing modes can be used to address one bit or a pair of bits of a specific register
- The direct addressing mode uses a bit offset to access a particular register's bit
- The offset is the number of bits counting from the least significant bit

Circular Addressing Mode

- Circular addressing mode provides an efficient method for accessing data buffers continuously without having to reset the data pointers
- After accessing data, data buffer pointer is updated in a modulo fashion
- When the pointer reaches the end of the buffer, it will wrap back to the beginning of the buffer for the next iteration

Circular Addressing Mode [2]

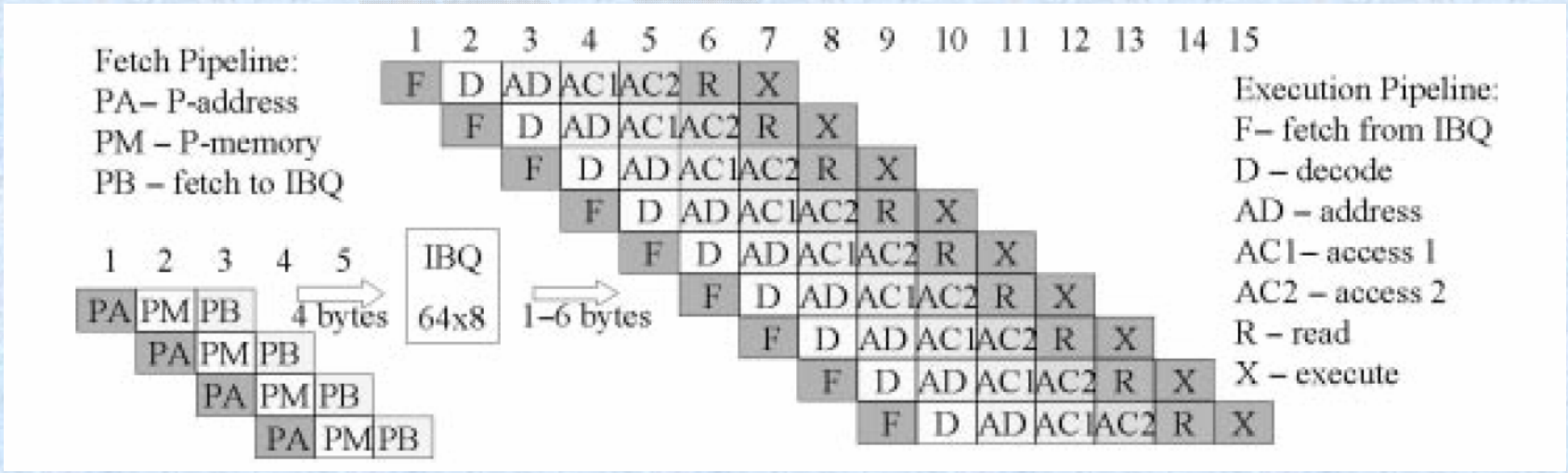
- Procedure
  - Initialise most significant 7-bit extended auxiliary register to select data page
  - Initialise 16-bit circular pointer to point to memory location within the buffer
  - Initialise 16-bit circular buffer starting address associated with the auxiliary registers
  - Initialise the data buffer size register
  - Enable the circular buffer configuration

Pipeline

- Pipeline technique has been widely used by many DSP manufacturers to improve processor performance
- Pipeline execution breaks sequence of operations into smaller segments and executes smaller pieces in parallel
- This is done to reduce overall execution time

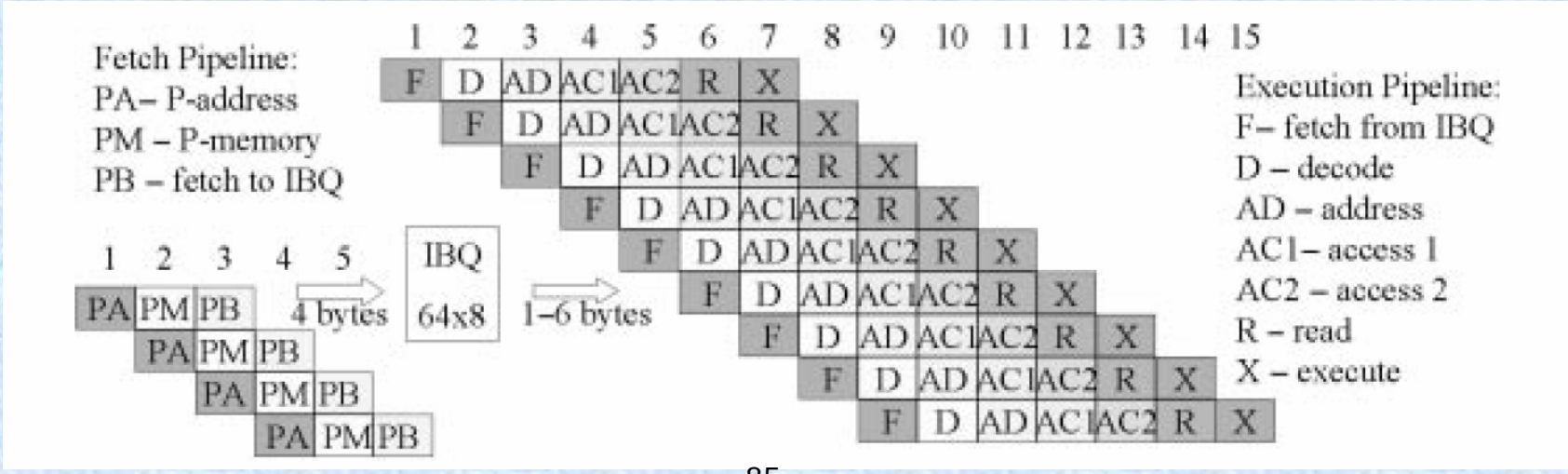
Pipeline [2]

- Separated by the instruction buffer unit, pipeline operation is divided into two independent pipelines – program fetch pipeline and program execution pipeline



Pipeline [3]

- Program fetch pipeline requires 3 clock cycles
  - Program address places on program-read address bus
  - C55x requires 1 clock cycle for address bus to be steady for memory read
  - 4 bytes of code are fetched via program data-read bus – code is placed in instruction buffer queue



Pipeline [4]

- Execution pipeline requires 7 stages
  - Fetch – instruction is fetched from Instruction Buffer Queue (can be 1 to 6 bytes)
  - Decode – Decode logic decodes Instruction or Instruction Pair under parallel operation – Instruction(s) are dispatched to program flow unit (PU), address flow unit (AU) or data computation unit (DU)
  - AU calculates data memory addresses, modifying pointers if required or address for branching
  - 1st cycle to send address for read operation from data buses (BAB, CAB, DAB) or transfer operand to CPU via (CB), 2nd cycle allows for address lines to be steady for 1 cycle before reading can occur
  - Read – Data and operands transferred to CPU via (CB) – AU generates address for operand write and sends address to data-write address buses (EAB, FAB)
  - Execute – Data processing is done in this cycle using ALU storing operands via (EB,FB)

Pipeline [5]

- Execution pipeline is full after 7 cycles and every cycle that follows will complete an instruction
- If pipeline is always full, technique increases 7 times the processing speed
- Pipeline problems are discussed later on

Parallel Execution

- Parallelism possible due to dual MAC and separate PU, AU and DU
- 2 parallel processing types
  - Implied using built-in instructions – use symbols ‘: :’ to separate instructions that can be parallelised
  - Explicit using user-built instructions – use symbols ‘| |’ to indicate pair of parallel instructions

Parallel Execution [2]

- Restrictions
  - Only two instructions can be parallelised and each cannot exceed 6 bytes length
  - Not all instructions can be used for parallel operation
  - When addressing memory space, only indirect addressing is allowed
  - Parallelism is allowed between and within execution units and there cannot be hardware resources conflicts

Concepts

Dr. Edward Gatt

Pipeline

- To speed execution, processors implement a pipeline (or *prefetch queue*)
- This is because many CPU instructions are fairly complex, taking many clock cycles to perform
- Normal CPUs, when executing multiple-clock instructions, the bus is normally idle
- In processors with pipeline, bus logic goes ahead and gets the next few instructions in preparation for execution

Pipeline [2]

- Pipeline architecture keeps CPU execution speed from being bogged down by slow memory
- While the CPU is executing multiple clock instructions, pipeline used this time to fill up with instructions
- The average rate of instruction execution cannot exceed the memory speed or else the pipeline can never get ahead of CPU

Pipeline [3]

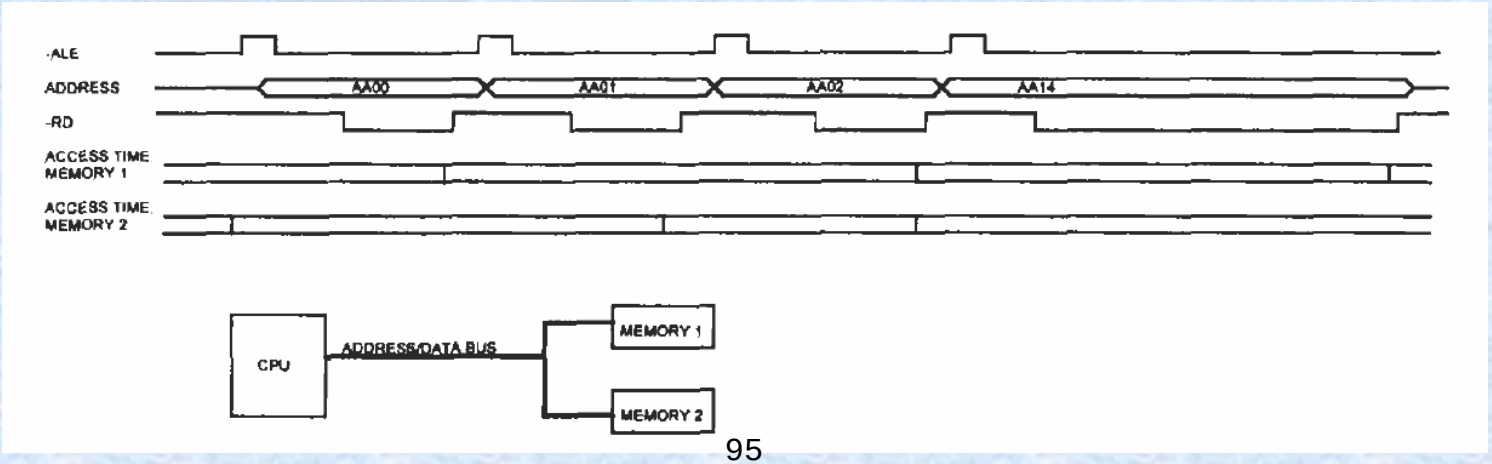
- Drawback to pipeline architecture is that, if a branch instruction is executed, all pre-fetched instructions must be discarded and the pipeline refilled from the new address
- To improve on this some processors fetch and partially decode the instructions in the pipeline and if decoding logic detects branch instructions, the pipeline will begin to fetch instructions from the new address in anticipation of the branch being taken

Pipeline [4]

- If the branch is conditional and not taken, the new instructions will need to be discarded and pre-fetching resumes from the address following the branch instruction
- Limitation of the system occurs with indirect addressing when the address is contained in a register and the register contents will depend on an instruction still in the pipeline – pipeline logic cannot pre-fetch data because the destination address is not known

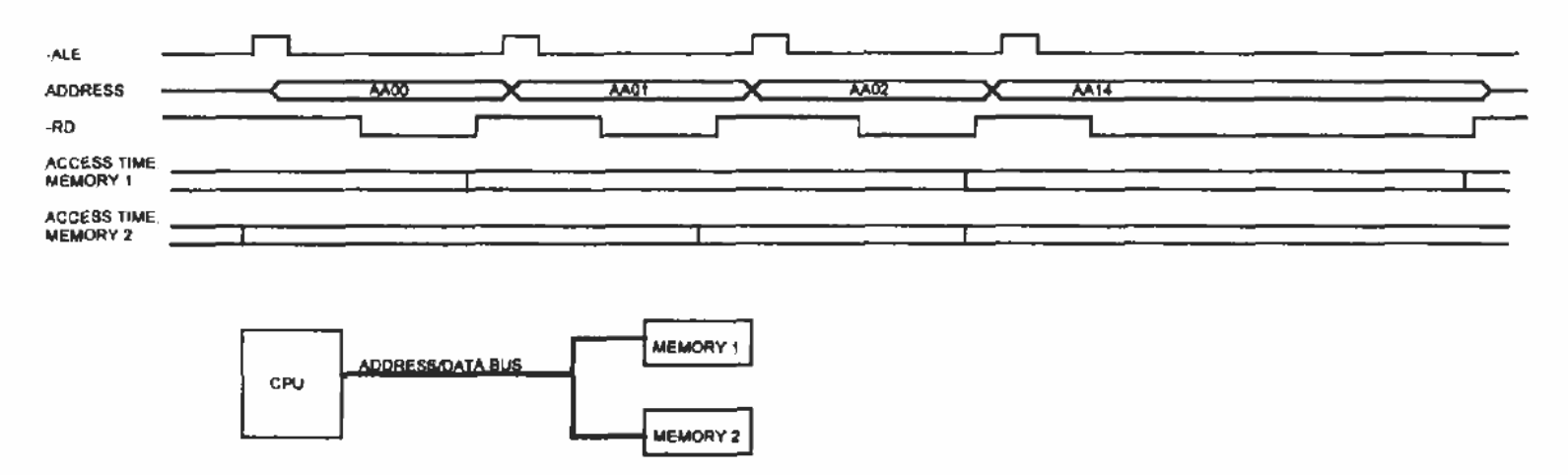
Interleaving

- Interleaving is used to allow a fast CPU to access slower memory without wait states
- Suppose a processor is accessing two memories – each having an access time longer than the bus cycle time
- Ordinarily, this would require the insertion of wait states
- However, if each memory is accessed on every other cycle, the two memories together can keep with the CPU
- Each memory access starts in a cycle when the other memory is being read, e.g. Memory 1 accessed on every even-numbered address and Memory 2 accessed on odd-numbered addresses



Interleaving [2]

- Interleaving works only as long as the processor executes sequential address cycles
- Access time for 1 memory device starts in the bus cycle for the other device – therefore next address for each device must be predictable
- e.g. CPU accessing addresses AA00, AA01, AA02, AA14…
- For AA14, memory access cannot be interleaved as the new address could not be predicted and so wait states must be inserted so that the memory can catch up with the CPU – as seen in the Figure below



Interleaving [3]

- Interleaving is performed in many microprocessor designs when interfacing to slower peripherals
- When microprocessor wants to read an ADC, it could start the ADC conversion and wait until conversion is complete -> CPU wastes time
- Solution -> CPU can start conversion – go away and perform other jobs – CPU returns for result and starts next conversion..

I/O Operations

- Three techniques for I/O operations:
  - Programmed I/O
  - Interrupt Driven I/O
  - Direct Memory Access

Programmed I/O

- With programmed I/O, data is exchanged between the CPU and the I/O modules
- CPU executes a program that gives it direct control of the I/O operation, including sensing the device status, sending a read or write command and transferring the data
- When CPU issues a command to I/O module, it must wait until the I/O operation is complete
- If CPU is faster than the I/O module, this is wasteful of CPU time

Interrupt Driven I/O

- CPU issues an I/O command, continues to execute other instructions and is interrupted by the I/O module when it has completed its work
- Interrupt Processing
  - Device issues an interrupt signal to the processor
  - Processor finishes execution of current instruction before responding to interrupt
  - Processor tests for interrupt, determines that there is one and sends acknowledge signal to device. Acknowledge allows device to remove interrupt signal

Interrupt Driven I/O [2]

- Interrupt Processing
  - Processor needs to prepare to transfer control to interrupt routine. It saves the information needed to resume the current program at point of interrupt. Minimum information required is status of processor and location of next instruction to be executed -> this value is pushed to system control stack
  - Processor loads PC with address of interrupt-handling program. If there is more than one interrupt-handling routine, processor must determine which to invoke

Interrupt Driven I/O [3]

- Interrupt Processing
  - Interrupt handler proceeds to process the interrupt – examines the status information related to I/O operation or other event causing the interrupt, plus additionally sending commands and acknowledgements to I/O device
  - When interrupt processing is complete, saved register values are retrieved from stack and restored to registers to continue previous program execution

Interrupt Driven I/O [4]

- Design Issues
  - How does CPU determine which device issued the interrupt (assuming multiple I/O modules)
  - If multiple interrupts occur, how does the CPU decide which one to process

Interrupt Driven I/O [5]

- Solutions
  - Multiple Interrupt Lines
  - Software Poll
  - Daisy Chain (Hardware Poll, Vectored)
  - Bus Arbitration (Vectored)

Multiple Interrupt Lines

- Simple approach – each device has its own interrupt line
- It is impractical to dedicate more than a few bus lines or CPU pins to interrupt lines
- Consequently, even if multiple lines are used, each line will have a number of I/O modules attached to it

Software Poll

- When CPU detects an interrupt, it branches to interrupt service routine whose job is to poll each I/O module to determine which module caused the interrupt – CPU raises address of particular I/O module on address line and I/O module responds positively if it set the interrupt OR each I/O module could contain an addressable status register and CPU reads these registers to identify the I/O module
- Disadvantage: Time Consuming

Daisy Chain

- For interrupts, all I/O modules share a common interrupt request line
- Interrupt acknowledge line is daisy chained through the modules
- When CPU senses an interrupt, it sends out an interrupt acknowledgement, which propagates through a series of I/O modules until it gets to a requesting module
- Requesting module typically responds by placing a word on the data lines
- Word is referred to as vector and is address of I/O module or some identifier of the module

Bus Arbitration

- I/O module must first gain control of the bus before it can raise the interrupt request line
- Therefore, only one module can raise the line at a time
- When CPU detects the interrupt, it responds on the interrupt acknowledge line
- Requesting module then places its vector on the data lines

Direct Memory Access (DMA)

- Drawbacks of Interrupt Driven and Programmed I/O:
  - Both still require the intervention of the CPU to transfer data between memory and I/O module
  - I/O transfer rate is limited by the speed with which the CPU can test and service a device
  - CPU is tied up in managing an I/O transfer as normally a number of instructions must be executed for each I/O transfer

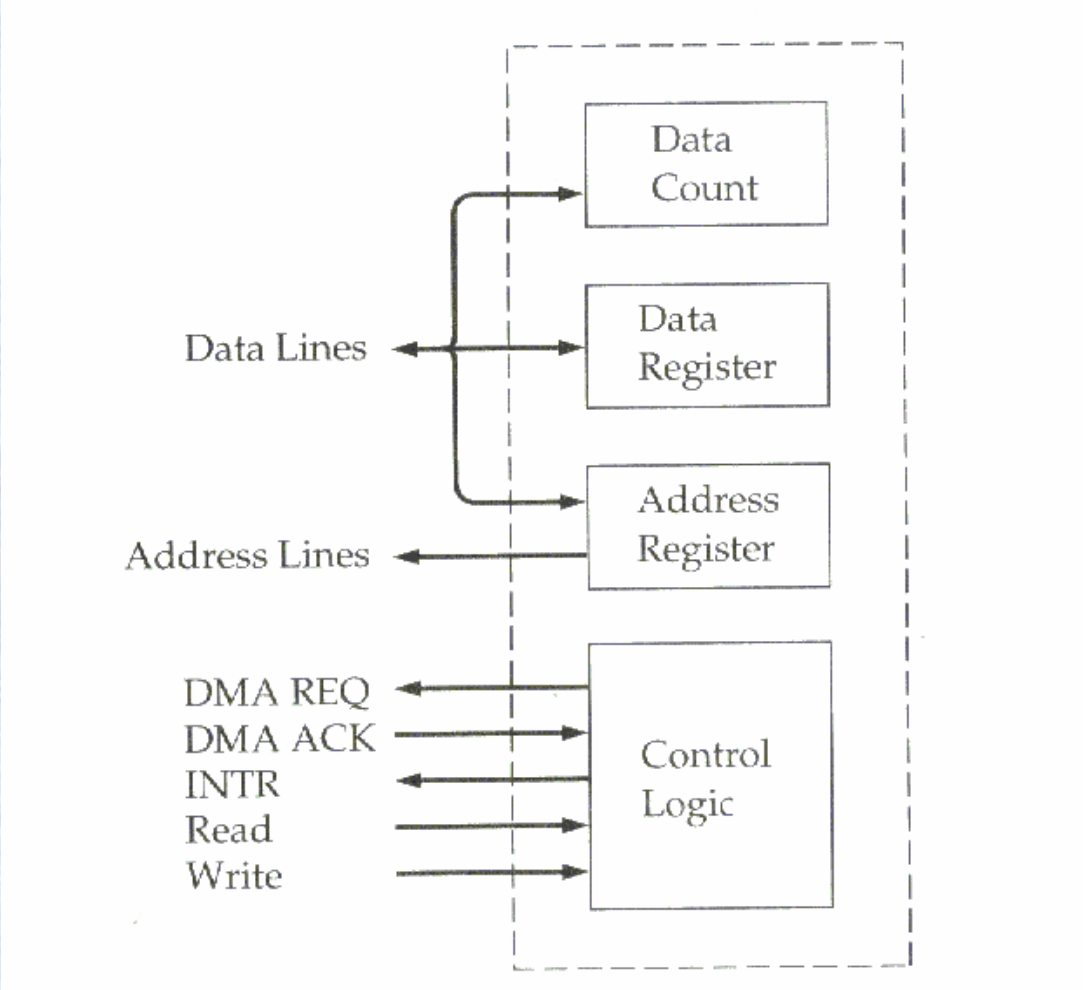
Direct Memory Access (DMA) [2]

- DMA involves an additional module on the system bus
- DMA module mimicks the CPU and takes control of the system from the CPU
- When CPU wishes to read or write a block of data, it issues a command to DMA module by sending following information
  - Whether read or write – Address of I/O device
  - Starting location in memory – No. of words to read or written

Direct Memory Access (DMA) [3]

- CPU continues with other work
- I/O operation delegated to DMA
- DMA module transfers entire block of data, one word at a time, directly to and from memory, without going through the CPU
- When transfer is complete, DMA module sends interrupt to CPU

Direct Memory Access (DMA) [4]



Direct Memory Access (DMA) [5]

- DMA module needs to take control of the bus in order to transfer data to and from memory
- DMA module must use bus only when CPU does not need it or it must force CPU to temporarily suspend operation – CPU suspended before it needs to use the bus
- For multiple-word I/O transfer, DMA is more efficient than interrupt-driven or programmed I/O

DSP Implementation Considerations

Dr. Edward Gatt

DSP

- The processing of digital signals can be described in terms of combinations of certain fundamental operations
- Operations include addition (or subtraction), multiplication and time-shift (or delay)

$$y(n) = x_1(n) + x_2(n)$$

Addition

```
mov @x1n, AC0   ; AC0 = x1(n)
add @x2n, AC0   ; AC0 = x1(n)+x2(n) = y(n)
```

DSP [2]

$$y(n) = \alpha x(n)$$

Multiplication

```
amov #alpha, XAR1       ; AR1 points to alpha (α)
amov #xn, XAR2          ; AR2 points to x(n)
mpy  *AR1, *AR2, AC0    ; AC0 = α* x(n) = y(n)
```

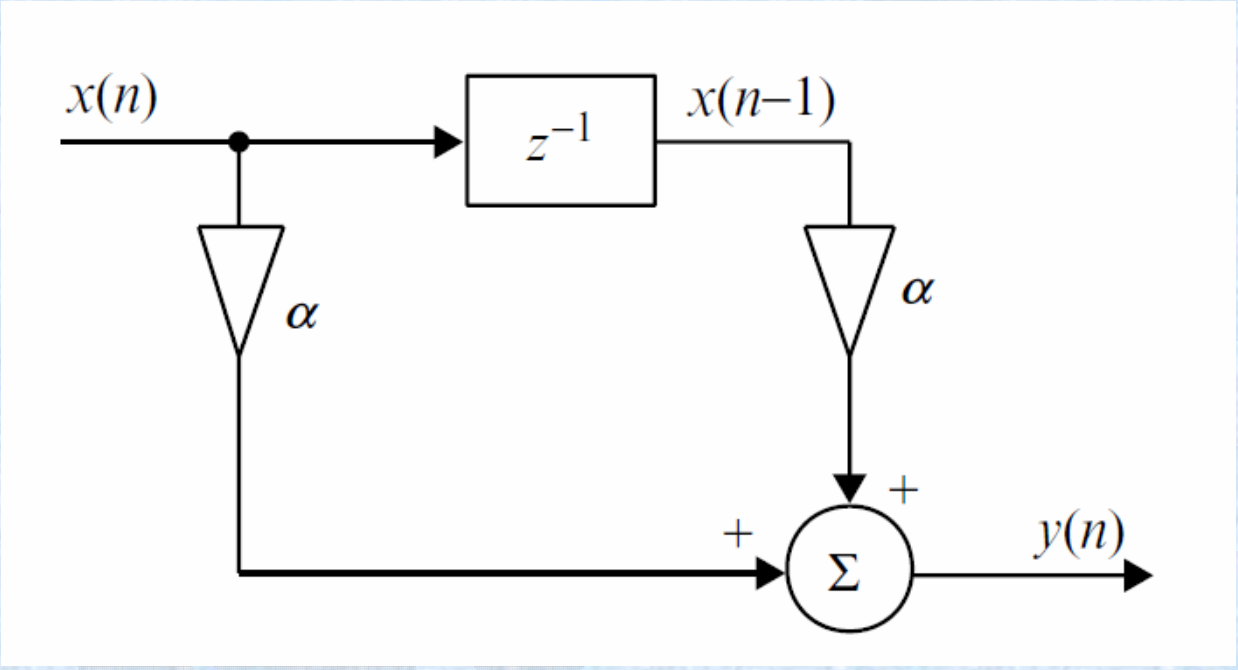
DSP [3]

$$y(n) = x(n-1)$$

Delay

```
amov #xn, XAR1   ; AR1 points to x(n)
delay *AR1       ; Contents of x(n) is copied to x(n−1)
```

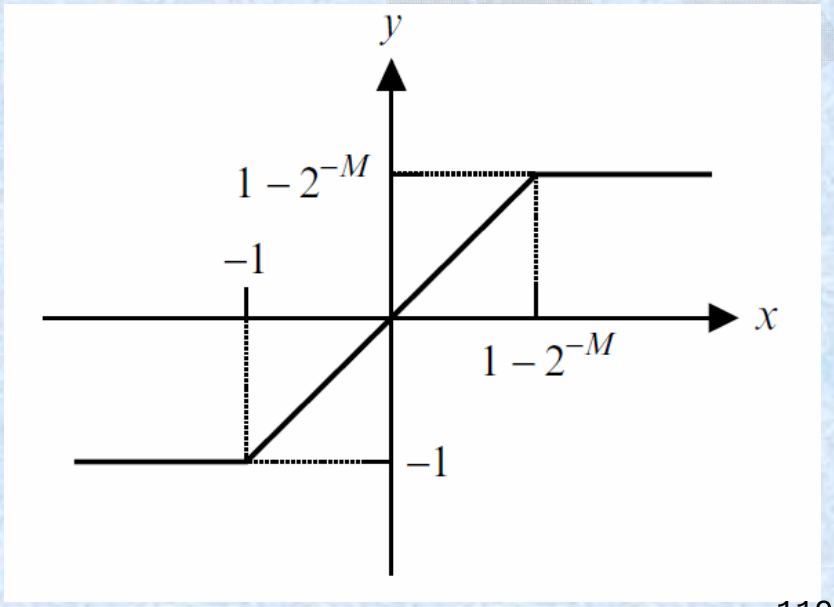
DSP Example




```
amov #alpha, XAR1       ; AR1 points to α
amov #temp, XAR2        ; AR2 points to temp
mov  *(x1n), AC0        ; AC0 = x1(n)
add  *(x2n), AC0        ; AC0 = x1(n)+x2(n)
mov  AC0, *AR2          ; Temp = x1(n)+x2(n), pointed by AR2
mpy  *AR1, *AR2, AC1    ; AC1 = α*[x1(n)+x2(n)]
```

Saturation Arithmetic

- Many DSPs have mechanisms to protect against overflow and indicate if it occurs
- Saturation prevents overflow by keeping the result to a maximum or minimum



$$y = \begin{cases} 1 - 2^{-M}, & x \geq 1 - 2^{-M} \\ x, & -1 \leq x < 1 \\ -1, & x < -1, \end{cases}$$

Synthesising Sine and Cos Functions Using Taylor Series

$$\cos(\theta) = 1 - \frac{1}{2!}\theta^2 + \frac{1}{4!}\theta^4 - \frac{1}{6!}\theta^6 + \cdots,$$

$$\sin(\theta) = \theta - \frac{1}{3!}\theta^3 + \frac{1}{5!}\theta^5 - \frac{1}{7!}\theta^7 + \cdots,$$

Multiplications and Additions (Subtractions) are easily implemented on DSP Processor