INSTITUTE OF AERONAUTICAL ENGINERRING

(AUTONOMOUS)

Dundigal, Hyderabad- 500 043

Presentation on
MICROCONTROLLERS AND PROGRAMMABLE DIGITAL SIGNAL PROCESSORS
(ECE)
M.TECH(ES) I- Semester
(AUTONOMOUS-R18)

Prepared by,
Mr. K.Chaitanya
(Assistant Professor)

SYSTEMS ARM CORTEX-M3 PROCESSOR

ARM CORTEX-M3 PROCESSOR

- RISC general purpose 32-bit microprocessor, released 2006
- Cortex-M3 differs from previous generations of ARM processors by defining a number of key peripherals as part of the core:
  - interrupt controller
  - system timer
  - debug and trace hardware (including external interfaces)
- This enables for real-time operating systems and hardware development tools such as debugger interfaces be common across the family of processors
- Various Cortex-M3 based microcontroller families differ significantly in terms of hardware **peripherals and memory**

ARM CORTEX-M3 PROCESSOR Cont..

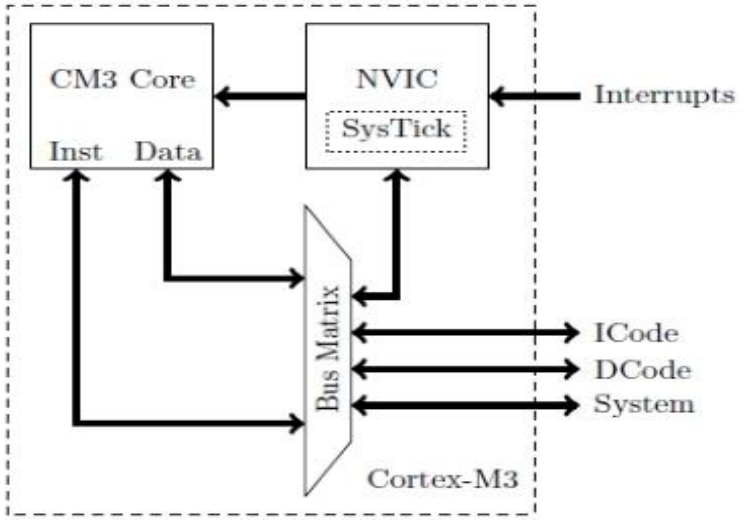
Greater performance efficiency: more work to be done without increasing the frequency or power requirements

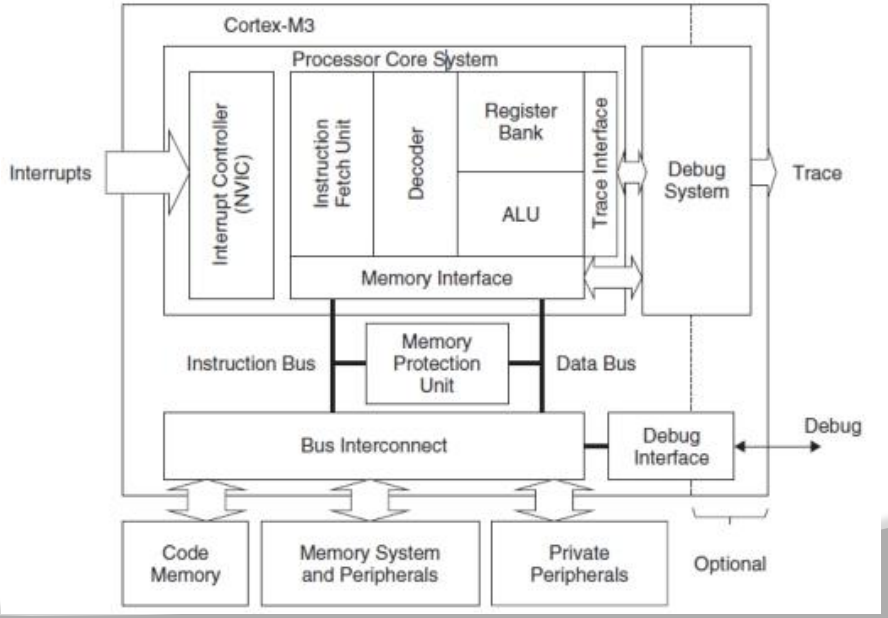
– Implements the new Thumb-2 instruction set architecture
- 70% more efficient per MHz than an ARM7TDMI-S processor executing Thumb instructions
- 35% more efficient than the ARM7TDMI-S processor executing ARM instructions for Dhrystone benchmark

ARM CORTEX-M3 PROCESSOR Cont ..

- **Low power consumption**: longer battery life, especially critical in portable products including wireless networking applications
- **Improved code density**: code fits in even the smallest memory footprints
- Core pipeline has 3 stages
  - Instruction Fetch
  - Instruction Decode
  - Instruction Execute

Simplified Cortex-M3 Architecture





Cortex-M3 Processor Architecture

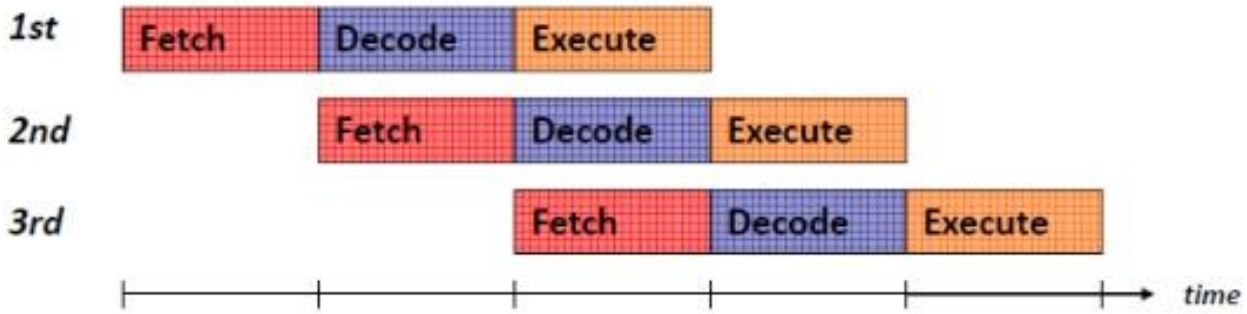
- Harvard architecture: it uses separate interfaces to fetch instructions (Inst) and (Data)
- Processor is not memory starved: it permits accessing data and instruction memories simultaneously
  - Only differentiates between instruction fetches and data accesses
- Interface between CM3 and manufacturer specific hardware is through three memory buses:
  - ICode, DCode, and System (for peripherals), which are defined to access different regions of memory

Cortex-M3 Processor Architecture cont..

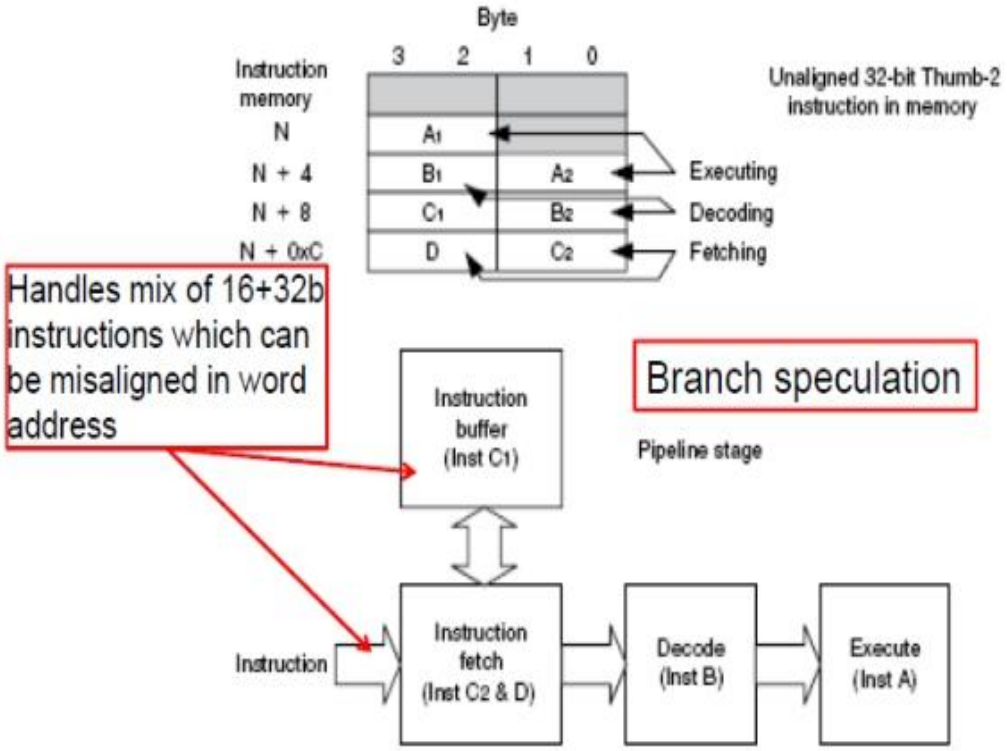
- Cortex-M3 is a load/store architecture with three basic types of instructions
- **register-to-register** operations for processing data
- **memory operations** which move data between memory and registers
- **control flow** operations enabling programming language control flow such as if and while statements and procedure calls

Cortex M3 pipelining

- The Cortex-M3 Uses the 3-stage pipeline for instruction executions
  - Fetch ⇒ Decode ⇒ Execute
  - Pipeline design allows effective throughput to increase to one instruction per clock cycle
  - Allows the next instruction to be fetched while still decoding or executing the previous instructions



Instruction Prefetch & Execution



Processor modes

- The ARM has seven basic operating modes:
  - Each mode has access to:
    - Its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode

	Mode	Description	
Exception modes	Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a low priority (normal) interrupt is raised	
	Abort	Used to handle memory access violations	
	Undef	Used to handle undefined instructions	
	System	Privileged mode using the same registers as User mode	
	User	Mode under which most Applications / OS tasks run	Unprivileged mode

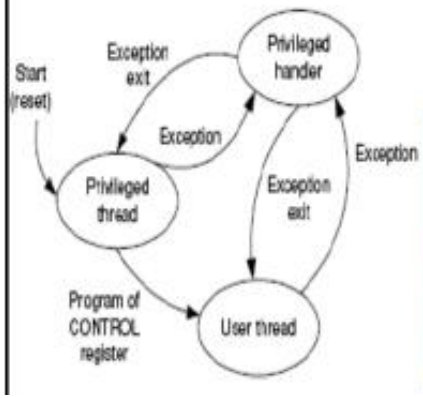
Operating modes

User mode:

- Normal program execution mode
- System resources unavailable
- Mode changed by exception only

Exception modes:

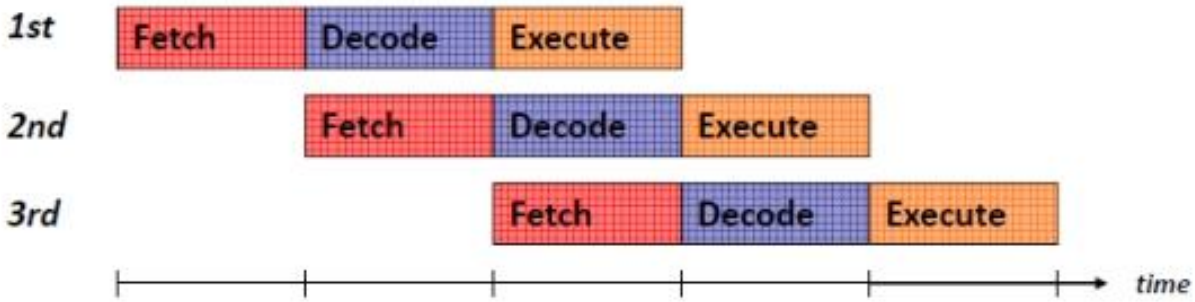
- Entered upon exception
- Full access to system resources
- Mode changed freely



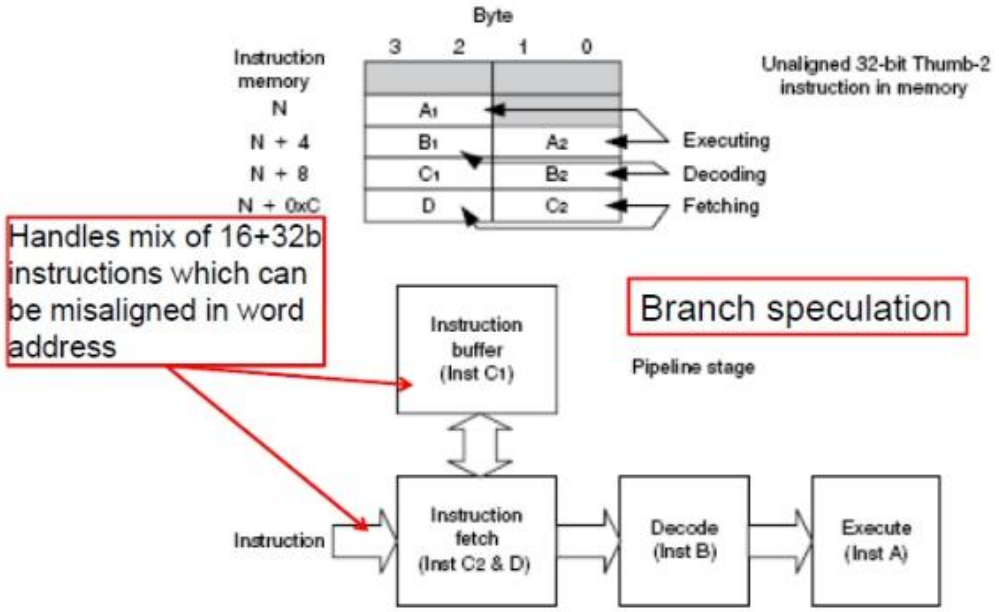
Modes (Thread out of reset)		Operations (privilege out of reset)	Stacks (Main out of reset)
	Handler - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

Cortex-M3 Pipeline

- The Cortex-M3 Uses the 3-stage pipeline for instruction executions
  - Fetch ⇒ Decode ⇒ Execute
  - Pipeline design allows effective throughput to increase to one instruction per clock cycle
  - Allows the next instruction to be fetched while still decoding or executing the previous instructions



Instruction Prefetch & Execution



Processor Modes

- The ARM has seven basic operating modes:
  - Each mode has access to:
    - Its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode

	Mode	Description	
Exception modes	Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a low priority (normal) interrupt is raised	
	Abort	Used to handle memory access violations	
	Undef	Used to handle undefined instructions	
	System	Privileged mode using the same registers as User mode	
	User	Mode under which most Applications / OS tasks run	Unprivileged mode

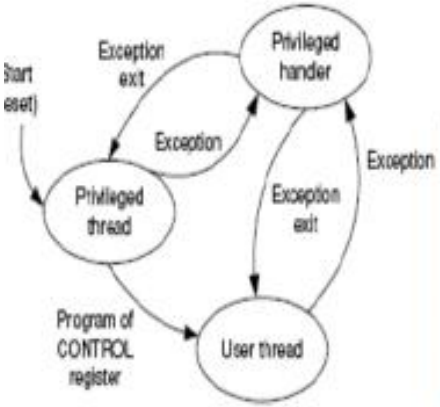
Operating Modes

User mode:

- Normal program execution mode
- System resources unavailable
- Mode changed by exception only

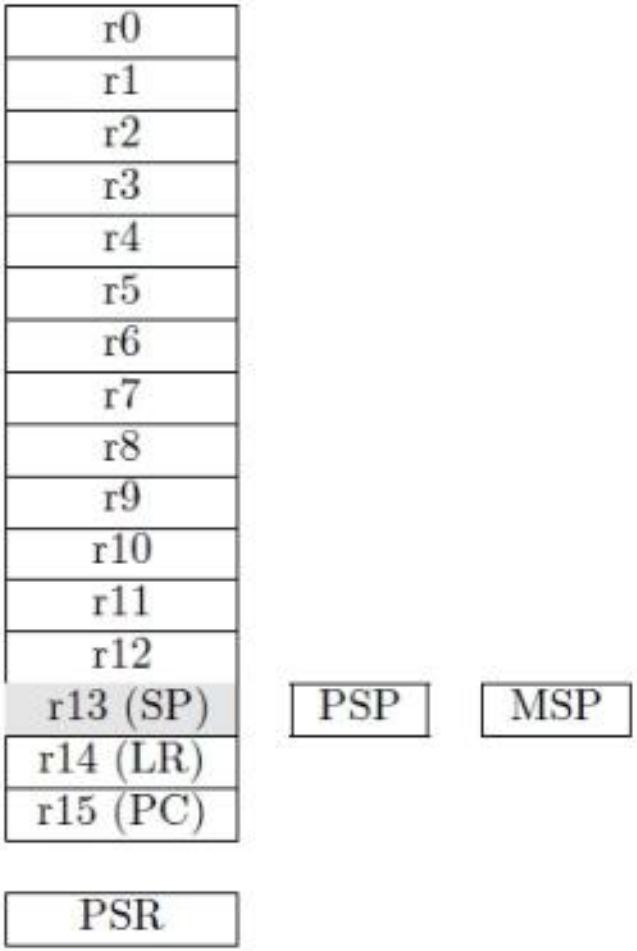
Exception modes:

- Entered upon exception
- Full access to system resources
- Mode changed freely



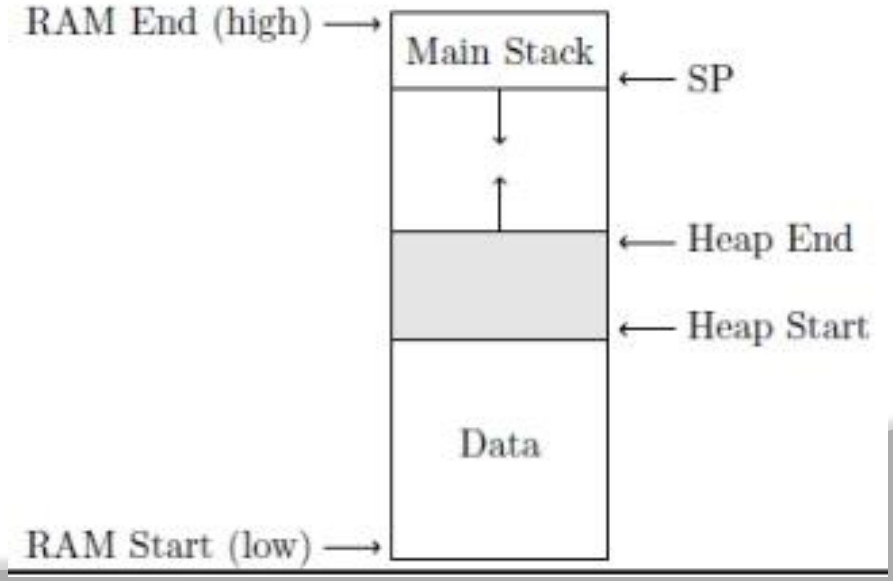
Modes (Thread out of reset)	Operations (privilege out of reset)	Stacks (Main out of reset)
Handler - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

Processor Register Set



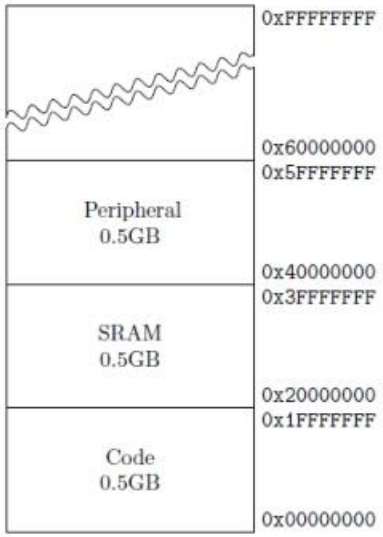
Program Memory Model

- RAM for an executing program is divided into three regions
  - Data in RAM are allocated during the link process and initialized by
    startup code at reset
  - The (optional) heap is managed at runtime by library code implementing functions such as the malloc and free which are part of the standard C library
  - The stack is managed at runtime by compiler generated code which generates per-procedure-call stack frames containing local variables and saved registers

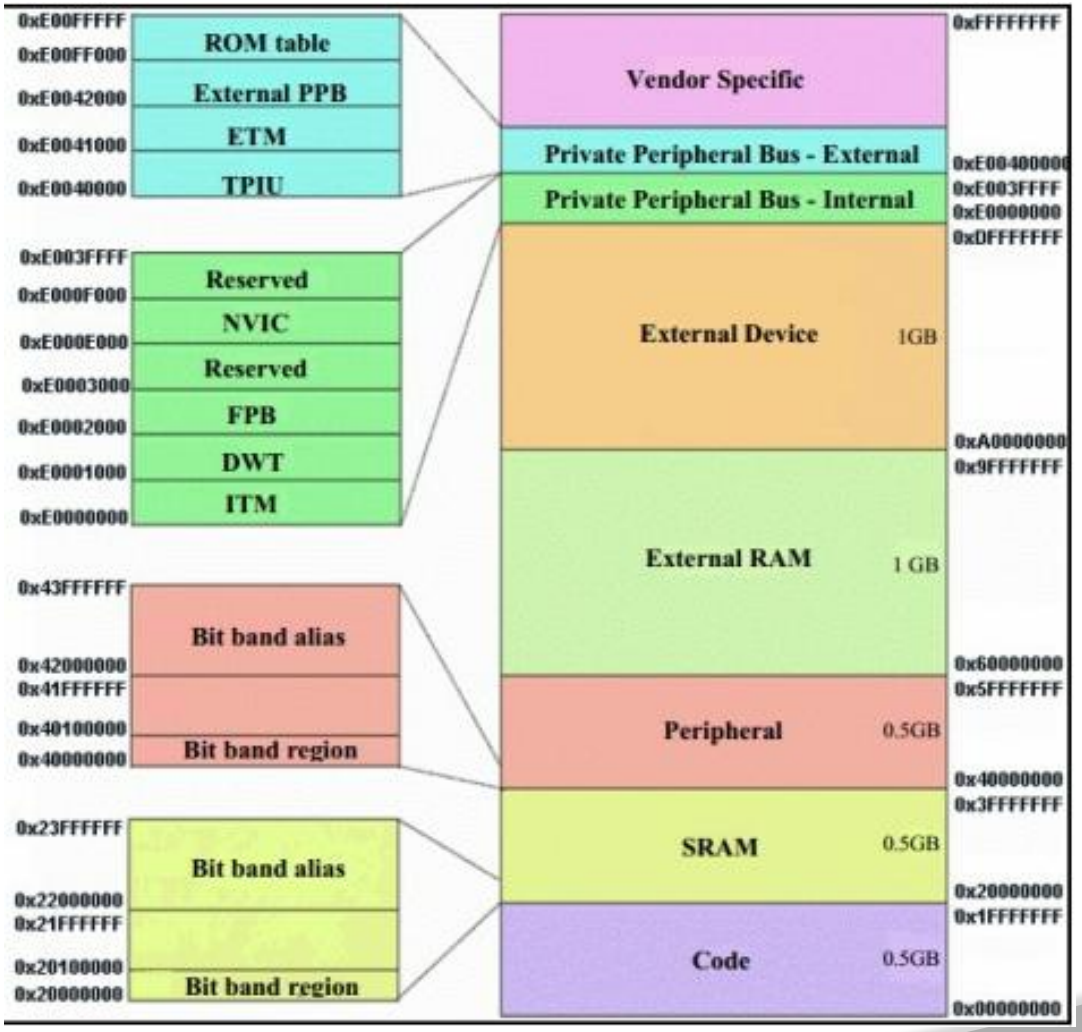


Cortex-M3 Memory Address Space

- ARM Cortex-M3 processor has a single 4 GB address space
- The **SRAM and Peripheral** areas are accessed through the System bus
- The **“Code” region** is accessed through the Icode (instructions) and Dcode (constant data) buses



Memory map


0xE00FFFFF
0xE00FF000
ROM table
0xE0042000
External PPB
0xE0041000
ETM
0xE0040000
TPIU
0xE003FFFF
0xE000F000
Reserved
NVIC
0xE000E000
Reserved
0xE0003000
FPB
0xE0002000
DWT
0xE0001000
ITM
0xE0000000
0x43FFFFFF
Bit band alias
0x42000000
0x41FFFFFF
0x40100000
Bit band region
0x40000000
0x23FFFFFF
Bit band alias
0x22000000
0x21FFFFFF
0x20100000
Bit band region
0x20000000
Vendor Specific
0xFFFFFFFF
Private Peripheral Bus - External
0xE00400000
Private Peripheral Bus - Internal
0xE003FFFF
0xE0000000
External Device
1GB
0xDFFFFFFF
0xA0000000
External RAM
1 GB
0x9FFFFFFF
0x60000000
Peripheral
0.5GB
0x5FFFFFFF
0x40000000
SRAM
0.5GB
0x3FFFFFFF
0x20000000
Code
0.5GB
0x1FFFFFFF
0x00000000

Instruction Set Architecture (ISA)

- Instruction set
  - Addressing modes
  - Word size
  - Data formats
  - Operating modes
  - Condition codes

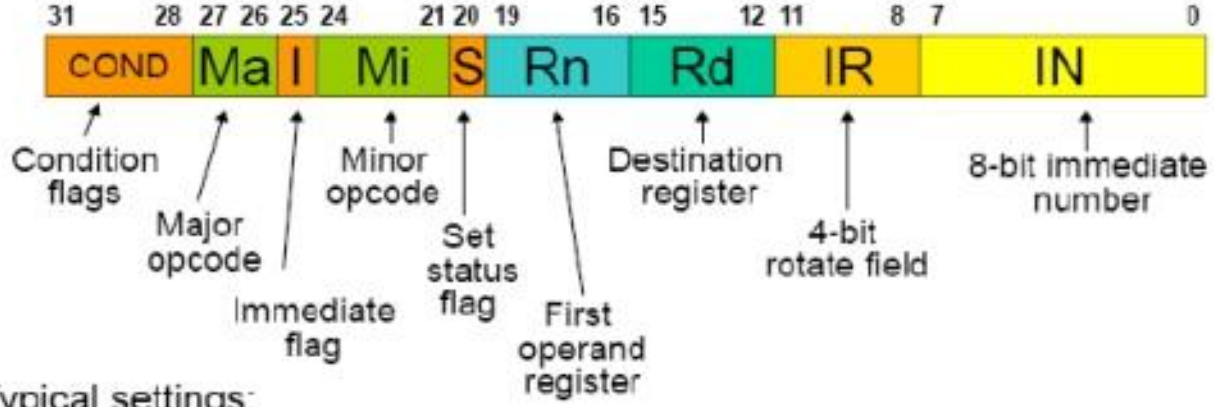
Traditional ARM instructions

- Fixed length of 32 bits
- Commonly take two or three operands
- Process data held in registers
- Shift & ALU operation in single clock cycle
- Access memory with load and store instructions only
  – Load/Store multiple register
- Can be extended to execute conditionally by adding the appropriate suffix
- Affect the CPSR status flags by adding the ‘S’ suffix to the instruction

32bit Instruction Encoding

Example: ADD instruction format

- ARM 32-bit encoding for ADD with immediate field



Typical settings:

Major opcode = 00 (this indicates data operation instructions)

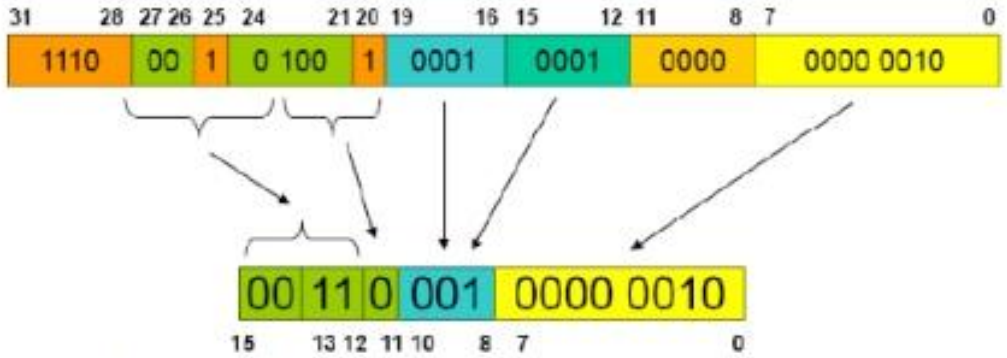
Minor opcode = 0100 (specifically, 100 ⇨ ADD instruction)

Immediate flag = 1 (immediate field in operand 2)

Set status flag = 1 (set carry flag after operation)

ARM and 16-bit Instruction Encoding

ARM 32-bit encoding: `ADDS r1, r1, #2`



- Equivalent 16-bit Thumb instruction: `ADD r1, #2`
  - No condition flag
  - No rotate field for the immediate number
  - Use 3-bit encoding for the register
  - Shorter opcode with implicit flag settings (e.g. the set status flag is always set)

Conditional Execution

- Each data processing instruction prefixed by condition code
- Result – smooth flow of instructions through pipeline
- 16 condition codes:

EQ	equal	MI	negative	HI	unsigned higher	GT	signed greater than
NE	not equal	PL	positive or zero	LS	unsigned lower or same	LE	signed less than or equal
CS	unsigned higher or same	VS	overflow	GE	signed greater than or equal	AL	always
CC	unsigned lower	VC	no overflow	LT	signed less than	NV	special purpose

Conditional Execution Cont..

- Every ARM (32 bit) instruction is conditionally executed.
- The top four bits are ANDed with the CPSR condition codes, If they do not matched the instruction is executed as NOP
- The AL condition is used to execute the instruction irrespective of the value of the condition code flags.
- By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using "S". Ex: SUBS r1,r1,#1
- Conditional Execution improves code density and performance by reducing the number of forward branch instructions.

Normal	Conditional
CMP r3,#0	CMP r3,#0
BEQ skip	ADDNE r0,r1,r2
ADD r0,r1,r2	
skip	

Conditional Execution and Flags

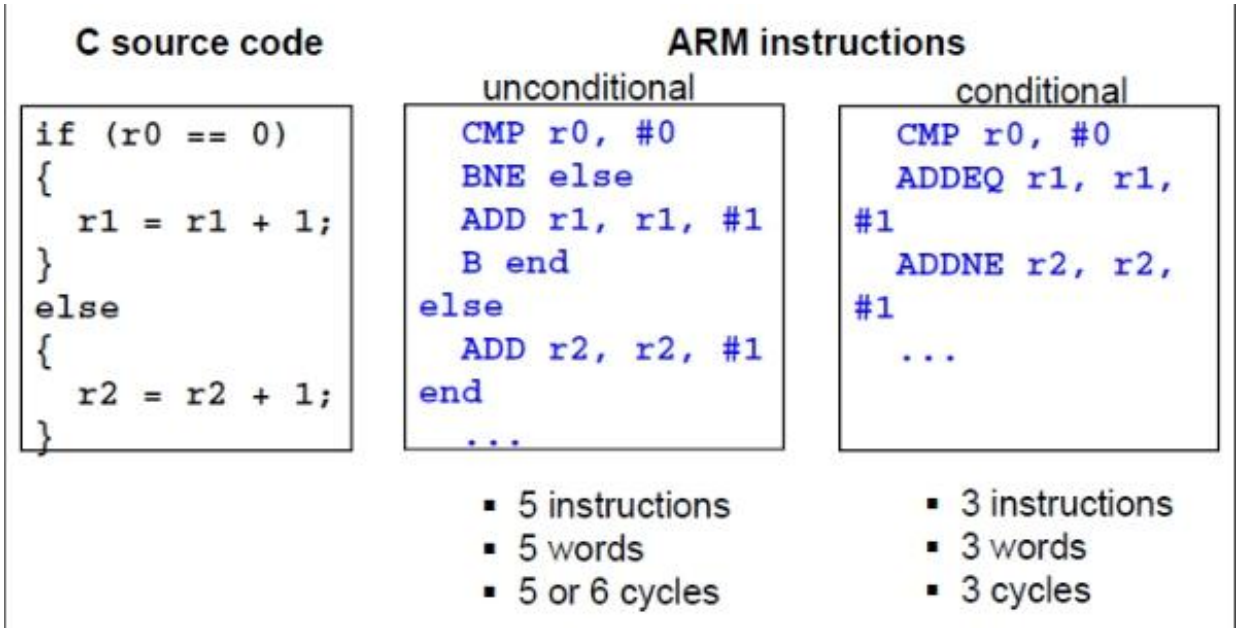
- ARM instructions can be made to execute conditionally by post-fixing them with the appropriate condition code
  - This can increase code density and increase performance by reducing the number of forward branches

```
CMP     r0, r1          ← r0 - r1, compare r0 with r1 and set flags
ADDGT   r2, r2, #1      ← if > r2=r2+1 flags remain unchanged
ADDLE   r3, r3, #1      ← if <= r3=r3+1 flags remain unchanged
```

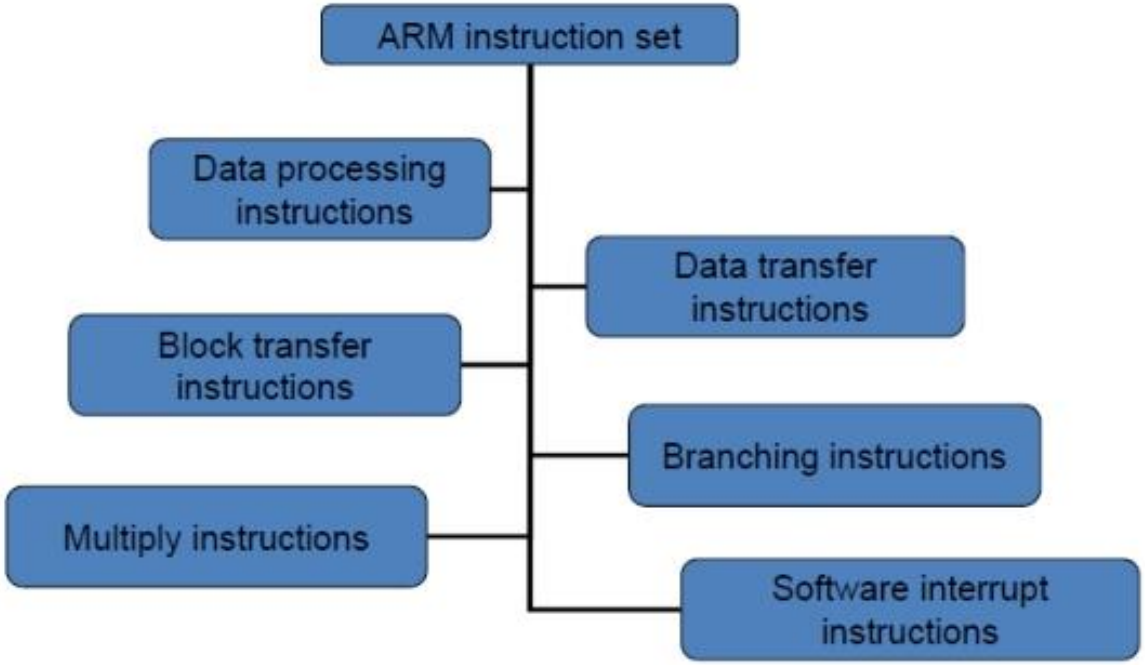
- By default, data processing instructions do not affect the condition flags but this can be achieved by post fixing the instruction (and any condition code) with an "S"

```
loop
    ADD     r2, r2, r3      ← r2=r2+r3
    SUBS    r1, r1, #0x01   ← decrement r1 and set flags
    BNE     loop            ← if Z flag clear then branch
```

Conditional execution examples



ARM Instruction Set



Data Processing Instructions

- Arithmetic and logical operations
- 3-address format:
  - Two 32-bit operands (op1 is register, op2 is register or immediate)
  - 32-bit result placed in a register
- Barrel shifter for op2 allows full 32-bit shift within instruction cycle

Data Processing Instructions

- Arithmetic operations:
  - ADD, ADDC, SUB, SUBC, RSB, RSC
- Bit-wise logical operations:
  - AND, EOR, ORR, BIC
- Register movement operations:
  - MOV, MVN
- Comparison operations:
  - TST, TEQ, CMP, CMN

Data Processing Instructions cont…

Conditional codes

+

Data processing instructions

+

Barrel shifter

=

Powerful tools for efficient coded programs

Data Processing Instructions

Multiply Instructions:

- Integer multiplication (32-bit result)
- Long integer multiplication (64-bit result)
- Built in Multiply Accumulate Unit (MAC)
- Multiply and accumulate instructions add product to running total

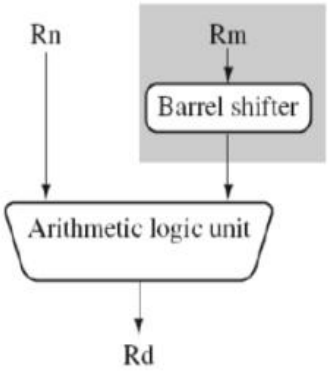
e.g.:

if (z==1) R1=R2+(R3*4)

compiles to

EQADDS R1,R2,R3, LSL #2

(SINGLE INSTRUCTION !)



Addressing Modes

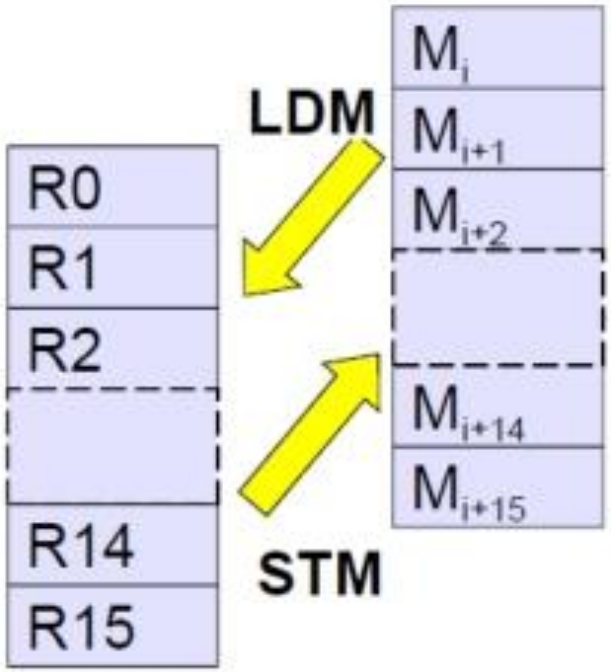
- Offset Addressing
  - Offset is added or subtracted from base register
  - Result used as effective address for memory access
    - [<Rn>, <offset>]
- Pre-indexed Addressing
  - Offset is applied to base register
  - Result used as effective address for memory access
  - Result written back into base register
  - [<Rn>, <offset>]!
- Post-indexed Addressing
  - The address from the base register is used as the EA
  - The offset is applied to the base and then written back – [<Rn>],offset>

<offset> options

- An immediate constant
  - #10
- An index register
  - <Rm>
- A shifted index register
  - <Rm>, LSL #<shift>

Block Transfer Instructions

- Load/Store Multiple instructions (*LDM/STM*)
- Whole register bank or a subset copied to memory or restored with single instruction

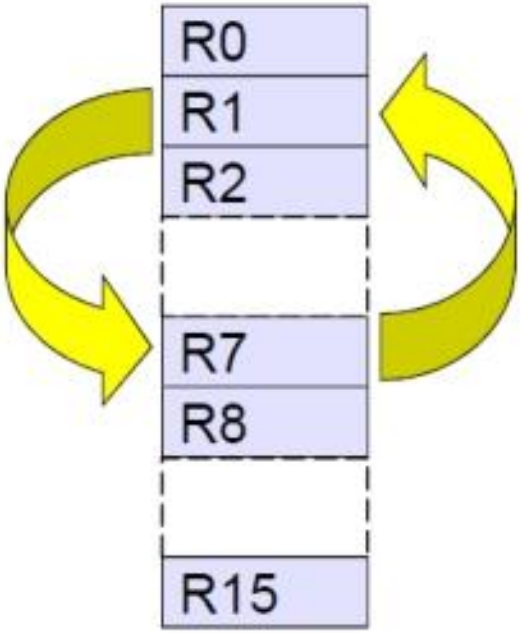


Swap Instruction

- Exchanges a word between registers
  - Two cycles

    but

  single atomic action
  - Support for RT semaphores



Unified Assembly Language

- UAL supports generation of either Thumb-2 or ARM instructions from the same source code
  - same syntax for both the Thumb code and ARM code
  - enable portability of code for different ARM processor families
- Interpretation of code type is based on the directive listed in the assembly file

- Example:
  - For GNU Assembler, the directive for UAL is

.syntax unified

  - For ARM assembler, the directive for UAL is

•THUMB

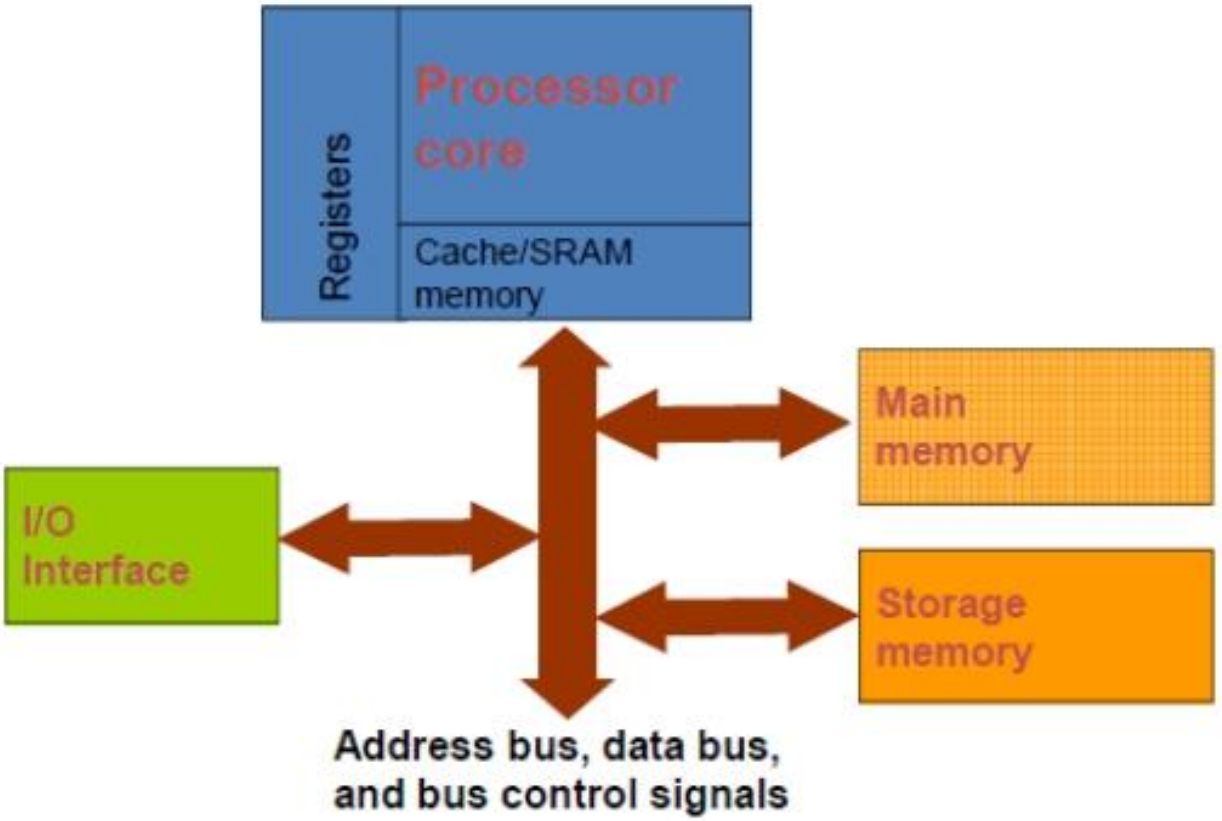
Example 1

```
ata:
        .byte 0x12, 20, 0x20, -1
func:
        mov r0, #0
        mov r4, #0
```

- movw r1, #:lower16:data
- movt r1, #:upper16:data
- top: ldrb r2, [r1],1

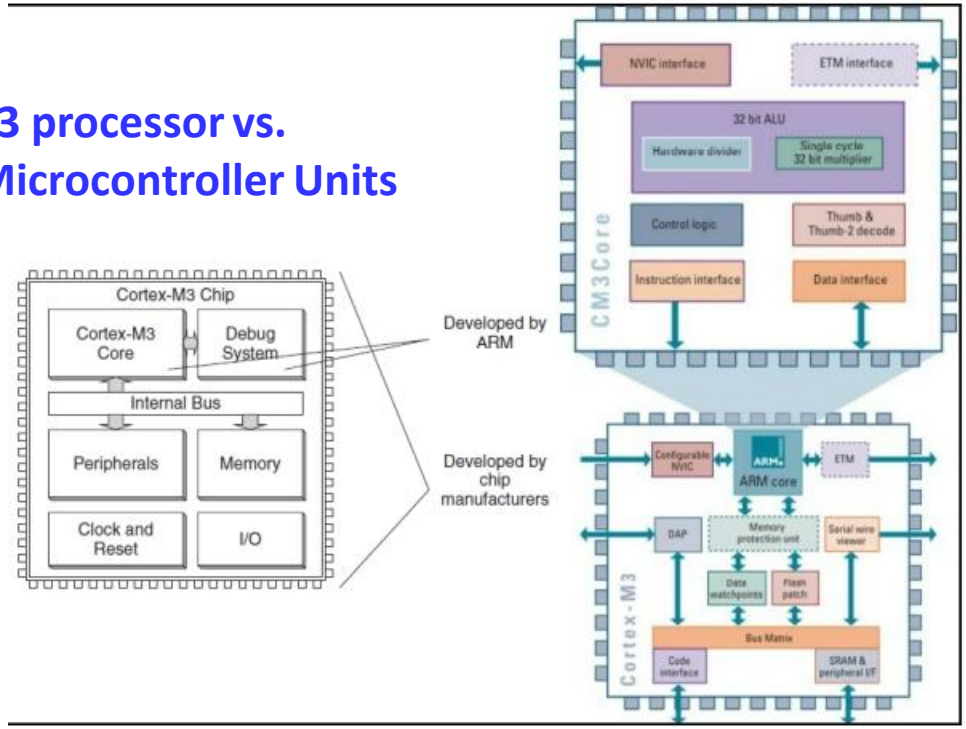
```
    add r4, r4, r2
    add r0, r0, #1
    cmp r0, #4
    bne top
```

Basic Processor Based System



Basic Processor Based System CONTD…

Cortex-M3 processor vs.
CM3-based Microcontroller Units



INSTITUTE OF AERONAUTICAL ENGINERRING

(AUTONOMOUS)

Dundigal, Hyderabad- 500 043

EXCEPTIONS AND INTERRUPTS

Interrupts and Exceptions

- An interrupt is usually defined as an event that alters the sequence of instructions executed by a processor.
- Such events correspond to electrical signals generated by hardware circuits both inside and outside the CPU chip.
- Interrupts are often divided into synchronous and asynchronous interrupts Synchronous interrupts are produced by the CPU control unit while executing instructions and are called synchronous because the control unit issues them only after terminating the execution of an instruction.
- Asynchronous interrupts are generated by other hardware devices at arbitrary times with respect to the CPU clock signals.

Interrupts

- Intel microprocessor manuals designate synchronous and asynchronous interrupts as exceptions and interrupts, respectively.

- We'll adopt this classification, although we'll occasionally use the term "interrupt signal" to designate both types together (synchronous as well as asynchronous).

Interrupts Cont..

Interrupt (a.k.a. exception or trap):

An event that causes the CPU to stop executing current program Begin executing a special piece of code Called an interrupt handler or interrupt service routine (ISR)

- Typically, the ISR does some work
- Then resumes the interrupted program

Interrupts Cont..

- Interrupts are really glorified procedure calls, except that they:
  - Can occur between any two instructions
  - Are “transparent” to the running program (usually)
  - Are not explicitly requested by the program (typically)
  - Call a procedure at an address determined by the type of interrupt, not the program

Interrupt Priority

What do we do if several interrupts arrive simultaneously?

NVIC allows priorities for (almost) every interrupt 3 fixed highest priorities, up to 256 programmable priorities 128 preemption levels Not all priorities have to be implemented by a vendor

Smart Fusion has 32 priority levels, i.e. 0x00, 0x08, … , 0xF8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented			Not implemented, read as zero				

Interrupt Priority

- Higher priority interrupts can pre-empt lower priorities
- Priority can be sub-divided into priority groups
  - Splits priority register into two halves, preempt priority & sub priority
  - Preempt priority: indicates if an interrupt can preempt another
  - Sub priority: used to determine which is served first if two interrupts of same group arrive concurrently

Interrupt Priority Cont..

- Interrupt priority level registers
  - ✓ Range: 0xE000E400 to 0xE000E4EF

Address	Name	Type	Reset Value	Description
0xE000E400	PRI_0	R/W	0 (8-bit)	Priority-level external interrupt #0
0xE000E401	PRI_1	R/W	0 (8-bit)	Priority-level external interrupt #1
…	–	–	–	–
0xE000E41F	PRI_31	R/W	0 (8-bit)	Priority-level external interrupt #31
…	–	–	–	–

Interrupt Vectors

- **Interrupt vector** is a pointer to an interrupt in memory
- Interrupt number is used to index the table
- **Interrupt vector table** holds pointers to all interrupts
- Table location may be fixed or placed in a known register

Interrupt Latency

- How quickly does the system respond to an interrupt?

Contributing Factors:

1. Maximum length of time when interrupts are disabled
2. Time required to execute higher priority interrupts
3. Time between interrupt event and running interrupt code
4. Time required to complete ISR code execution

Reducing Interrupt Latency

- **Make interrupt code short**
  - Reduces ISR execution time and time for higher priority interrupts
- **Reduce time during which interrupts are disabled**
  - Minimize size of critical regions

Interrupt Flags

- When an interrupt occurs, a **flag** bit is set in a register

TIFR0 – Timer/Counter Interrupt Flag Register
- Contains the flags for Output Compare and Overflow

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TOV0 – Indicates that timer 0 overflow occurred
OCF0A – Indicates that TCNT0 == OCR0A
OCF0B – Indicates that TCNT0 == OCR0B

- All flags are cleared when the interrupt is executed
- You should not need to access this register directly

Timer Counter Control Registers

- **TCCR0A** and **TCCR0B** control different aspects of timer function

Compare/Match Output Modes (COM0A1:0)

- **OC0A** is an output pin of the Atmega 2560
- Output comparison matching can drive the output pin
- Typically used to generate regular waveforms (like PWM)
- Can be used to synchronize system components
- We will not use this feature

Timer Counter Control Registers

Waveform Generation Modes (WGM2:0)

- Specify properties of PWM signals generated
- Frequency, width, etc.
- We will not use this feature

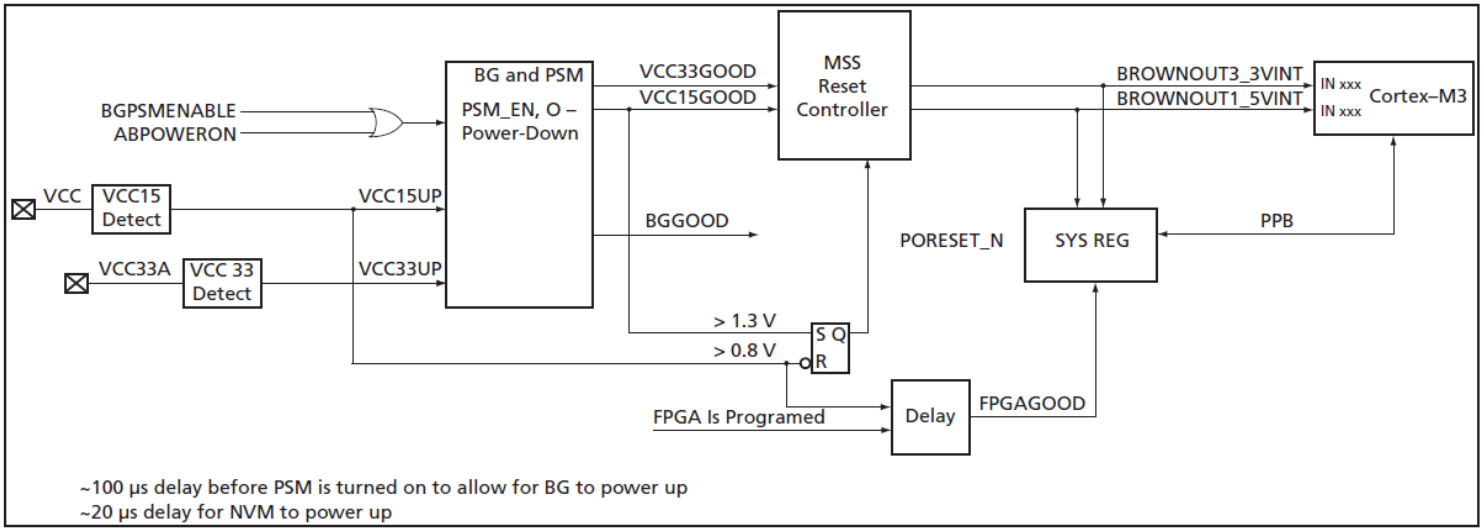
Force Output Compare (FOC0A, FOC0B)

- Forces the output compare to evaluate true, even if it didn't occur
- As if TCNT0 == OCR0A or TCNT0 == OCR0B
- Used to alter waveform on OCOA or OCOB pins
- We will not use this feature

Tail chaining

- When new exception occurs
- But CPU handling another exception of same/higher priority
- New exception will enter pending state
- But will be executed before register unstacking
- Saving unnecessary unstacking/stacking operations
- Can reenter hander in as little as 6 cycles

Example of Complexity: The Reset Interrupt



1) No power
2) System is held in RESET as long as VCC15 < 0.8V
    a) In reset: registers forced to default
    b) RC-Osc begins to oscillate
    c) MSS_CCC drives RC-Osc/4 into FCLK
    d) PORESET_N is held low
3) Once VCC15GOOD, PORESET_N goes high
    a) MSS reads from eNVM address 0x0 and 0x4

Nested Vectored Interrupt Controller (NVIC)

Hardware unit that coordinates among interrupts from multiple sources

- Define priority level of each interrupt source (**`NVIC_PRIx_R`** registers)
- Separate enable flag for each interrupt source (**`NVIC_EN0_R`** and **`NVIC_EN1_R`**)

Interrupt *does not* set I bit

- Higher priority interrupts can interrupt lower priority ones

NVIC Interrupt Enable Registers

Two enable registers –

`NVIC_EN0_R` and `NVIC_EN1_R`

Each 32-bit register has a single enable bit for a particular device
`NVIC_EN0_R` control the IRQ numbers 0 to 31 (interrupt numbers 16 – 47)
`NVIC_EN1_R` control the IRQ numbers 32 to 47 (interrupt numbers 48 – 63)

Interrupt Service Routine (ISR)

Things you must do in every interrupt service routine

- Acknowledge
- clear flag that requested the interrupt
- SysTick is exception
- automatic acknowledge
- Maintain contents of R4-R11 (AAPCS) Communicate via shared global variables

SYSTICK Timer

- System Timer zCortex-M3 includes an integrated system timer, SysTick is Provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter is an RTOS tick timer which fires at a programmable rate (for example, 100 Hz)

- Invokes a SysTick handler routine is ahigh-speed alarm timer using the system clock is a variable rate alarm or signal timer is a simple counter used to measure time to completion and time used.

SYSTICK Timer Contd..

- Functional Description is The timer consists of three registers: SysTick Control and Status Register: a control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status is SysTick Reload Value Register.

- The reload value for the counter, used to provide the counter's wrap value is SysTick Current Value Register: the current value of the counter is Note: the SysTick Calibration Value Register, is not implemented in the Stellaris devices.

SYSTICK Timer

- Writing to the SysTick Current Value register clears the register and the COUNTFLAG status bit On a read from the SysTick Current Value register, the current value is the value of the register at the time the register is accessed If the core is in debug state (halted).
- the counter does not decrementζ When the counter reaches zero, the COUNTFLAG status bit is set (clears on reads) ζFunctional Description is When enabled, the timer counts down on each clock from the reload value to zero Reloads (wraps) to the value in the SysTick Reload Value register on the next clock edge, then decrements on subsequent clocks Clearing the SysTick Reload Value register disables the counter on the next wrap.

SysTick Control and Status Register

- SysTick Reload Value Register isThe start value N can be between 1 and 0x00FF.FFFF, firing every N+1 clock.

- For example, if a tick interrupt is required every 100 clock pulses, 99 must be written into the RELOAD field SysTick Current Value Register is The SysTick Current Value Register contains the current value of the counter.

SysTick Control and Status Register Cont..

- General-Purpose Timers zProgrammable timers can be used to count or time external events that drive the Timer input pins zThe Stellaris General-Purpose Timer Module (GPTM) contains four GPTM blocks zEach GPTM block provides two 16-bit timers/counters zcan be configured to operate independently as timers or event counters zor configured to operate as one 32-bit timer or one 32- bit Real-Time Clock (RTC).

EXCEPTIONS

- The 80x86 microprocessors issue roughly 20 different exceptions (The exact number depends on the processor model.) The values from 20 to 31 are reserved by Intel for future development.
- Each exception is handled by a specific exception handler, which usually sends a Unix signal to the process that caused the exception.
- The kernel must provide a dedicated exception handler for each exception type.
- For some exceptions, the CPU control unit also generates a hardware error code and pushes it on the Kernel Mode stack before starting the exception handler.

EXCEPTIONS Contd..

The following list gives the vector, the name, the type, and a brief description of the exceptions found in 80x86 processors.

0 - "Divide error" (fault): Raised when a program issues an integer division by 0.

1- "Debug" (trap or fault): Raised when the **TF flag of eflags** is set (quite useful to implement single-step execution of a debugged program) or when the address of an instruction or operand falls within the range of an active debug register .

EXCEPTIONS LIST

11 - "Segment not present" (fault): A reference was made to a segment not present in memory (one in which the Segment-Present flag of the Segment Descriptor was cleared).

12 - "Stack segment fault" (fault): The instruction attempted to exceed the stack segment limit, or the segment identified by ss is not present in memory.

13 - "General protection" (fault): One of the protection rules in the protected mode of the 80x86 has been violated.

EXCEPTIONS LIST Contd..

14 - "Page Fault" (fault): The addressed page is not present in memory, the corresponding Page Table entry is null, or a violation of the paging protection mechanism has occurred.

15 - Reserved by Intel

16 - "Floating-point error" (fault): The floating-point unit integrated into the CPU chip has signalled an error condition, such as numeric overflow or division by 0.

EXCEPTIONS LIST Contd..

17 - "Alignment check" (fault): The address of an operand is not correctly aligned (for instance, the address of a long integer is not a multiple of 4).

18 - "Machine check" (abort): A machine-check mechanism has detected a CPU or bus error.

19 - "SIMD floating point exception" (fault): The SSE or SSE2 unit integrated in the CPU chip has signalled an error condition on a floating-point operation

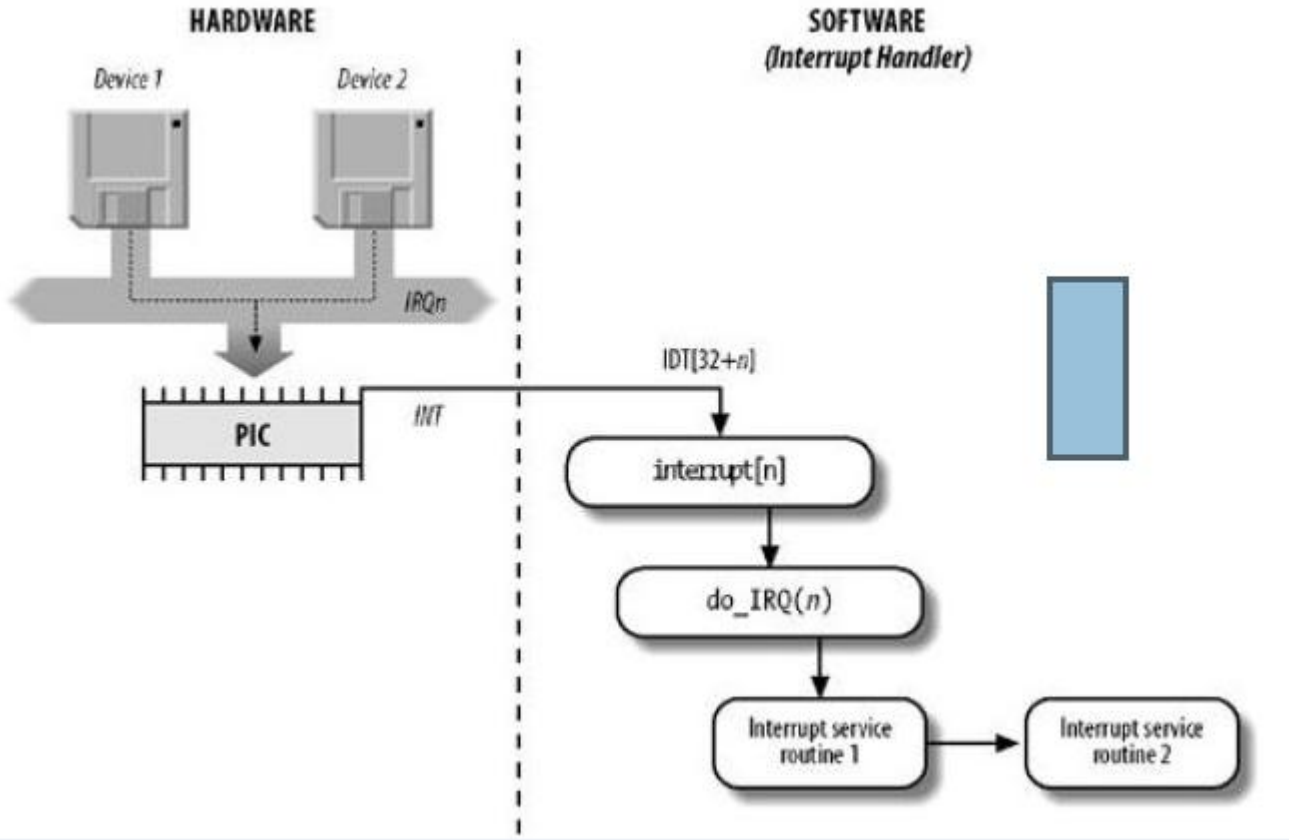
Table _ Signals sent by the Exception handlers

#	Exception	Exception handler	Signal
0	Divide error	divide_error( )	SIGFPE
1	Debug	debug( )	SIGTRAP
2	NMI	nmi( )	None
3	Breakpoint	int3( )	SIGTRAP
4	Overflow	overflow( )	SIGSEGV
5	Bounds check	bounds( )	SIGSEGV
6	Invalid opcode	invalid_op( )	SIGILL
7	Device not available	device_not_available( )	None
8	Double fault	doublefault_fn( )	None
9	Coprocessor segment overrun	coprocessor_segment_overrun( )	SIGFPE
10	Invalid TSS	invalid_TSS( )	SIGSEGV
11	Segment not present	segment_not_present( )	SIGBUS
12	Stack segment fault	stack_segment( )	SIGBUS
13	General protection	general_protection( )	SIGSEGV
14	Page Fault	page_fault( )	SIGSEGV
15	Intel-reserved	None	None
16	Floating-point error	coprocessor_error( )	SIGFPE
17	Alignment check	alignment_check( )	SIGBUS
18	Machine check	machine_check( )	None
19	SIMD floating point	simd_coprocessor_error( )	SIGFPE

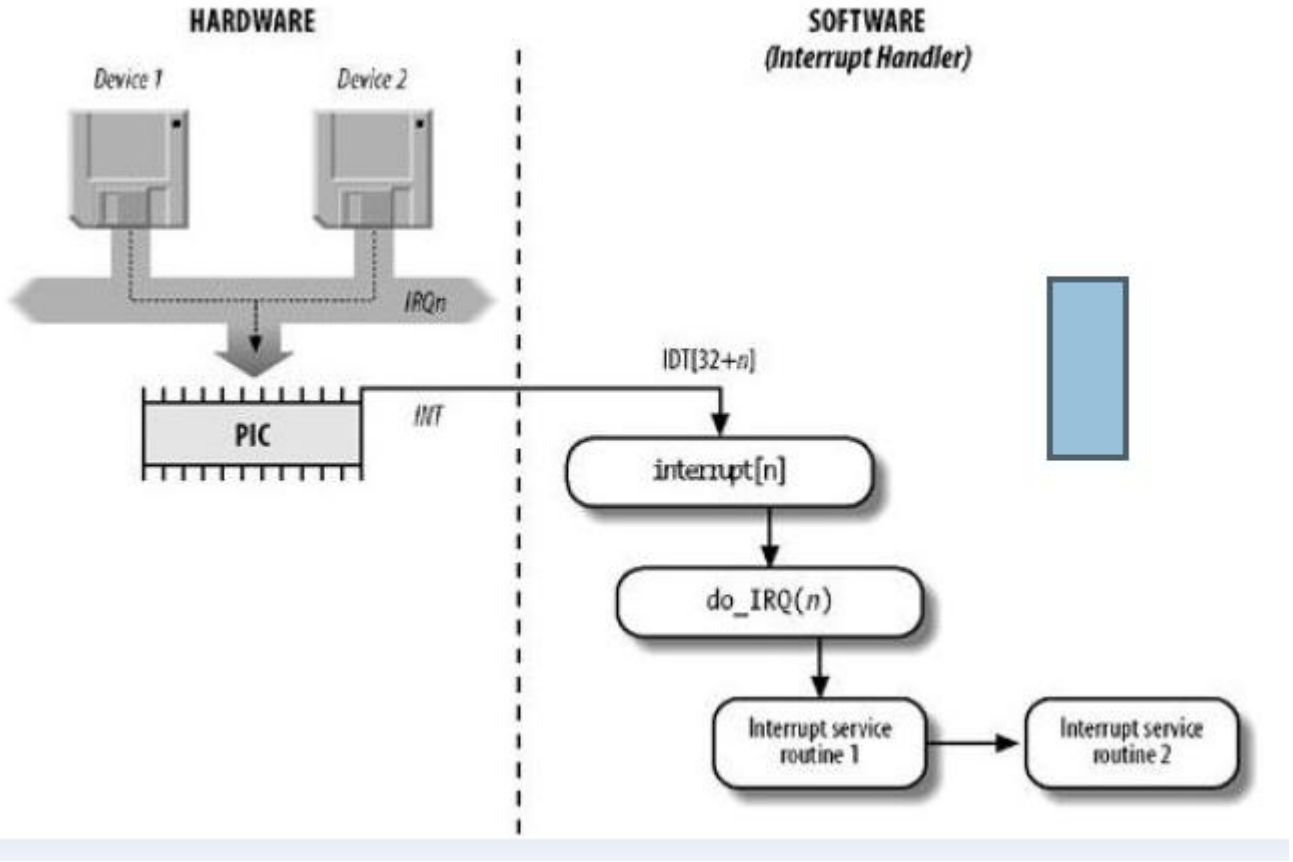
EXCEPTION HANDLERS

- Exception handlers have a standard structure consisting of three steps:
  - Save the contents of most registers in the Kernel Mode stack (this part is coded in assembly language).
  - Handle the exception by means of a high-level C function.
  - Exit from the handler by means of the ret_from_exception( ) function.
- To take advantage of exceptions, the IDT must be properly initialized with an exception handler function for each recognized exception.
- It is the job of the trap_init( ) function to insert the final values, the functions that handle the exceptions, into all IDT entries that refer to nonmaskable interrupts and exceptions.

I/O Interrupt handling



I/O Interrupt handling



LPC 17XX MICROCONTROLLER

INTERNAL MEMORY

- Memory management:

  - Independent transmit and receive buffers memory mapped to shared SRAM.

  - DMA managers with scatter/gather DMA and arrays of frame descriptors.

  - Memory traffic optimized by buffering and pre-fetching.

General purpose serial I/O

- Device pins that are not connected to a specific peripheral function are controlled by the GPIO registers.
- Pins may be dynamically configured as inputs or outputs.
- Separate registers allow setting or clearing any number of outputs simultaneously.
- The value of the output register may be read back as well as the current state of the port pins.

LPC17xx use accelerated GPIO functions:

a. GPIO registers are accessed through the AHB multilayer bus so that the fastest possible I/O timing can be achieved.
b. Mask registers allow treating sets of port bits as a group, leaving other bits unchanged.
c. All GPIO registers are byte and half-word addressable.
d. Entire port value can be written in one instruction.
e. Support for Cortex-M3 bit banding.
f. Support for use with the GPDMA controller.

Features:

- Arm Cortex-M3 processor, running at frequencies of up to 100 MHz (LPC1768/67/66/65/64/63) or of up to 120 MHz (LPC1769). A Memory Protection Unit (MPU) supporting eight regions is included.
- Arm Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512 kB on-chip flash programming memory. Enhanced flash memory accelerator enables high-speed 120 MHz operation with zero wait states.
- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip bootloader software.
- On-chip SRAM includes:
  - 32/16 kB of SRAM on the CPU with local code/data bus for high-performance CPU access.

Features Cont..

- Bit level set and clear registers allow a single instruction to set or clear any number of bits in one port.
- Direction control of individual bits.
- All I/O default to inputs after reset.
- Pull-up/pull-down resistor configuration and open-drain configuration can be programmed through the pin connect block for each GPIO pin.

TIMERS

Steps to Configure Timer:

a. Power On the Timer module by setting the appropriate bits in PCONP register.
b. Configure MCR to reset the TC and generate the interrupt whenever it matches MRx.
c. Set the pre-scalar value for 1us.
d. Update the MRx register with required delay in micro secs.
e. Configure TCR to enable the Counter for incrementing the TC.
f. Enable the required timer interrupt.
g. Configure the GPIO pins as output for blinking the LEDs.
h. Toggle the LEDs in ISR whenever the interrupt is generated.

LPC 176XX TIMER REGISTERS

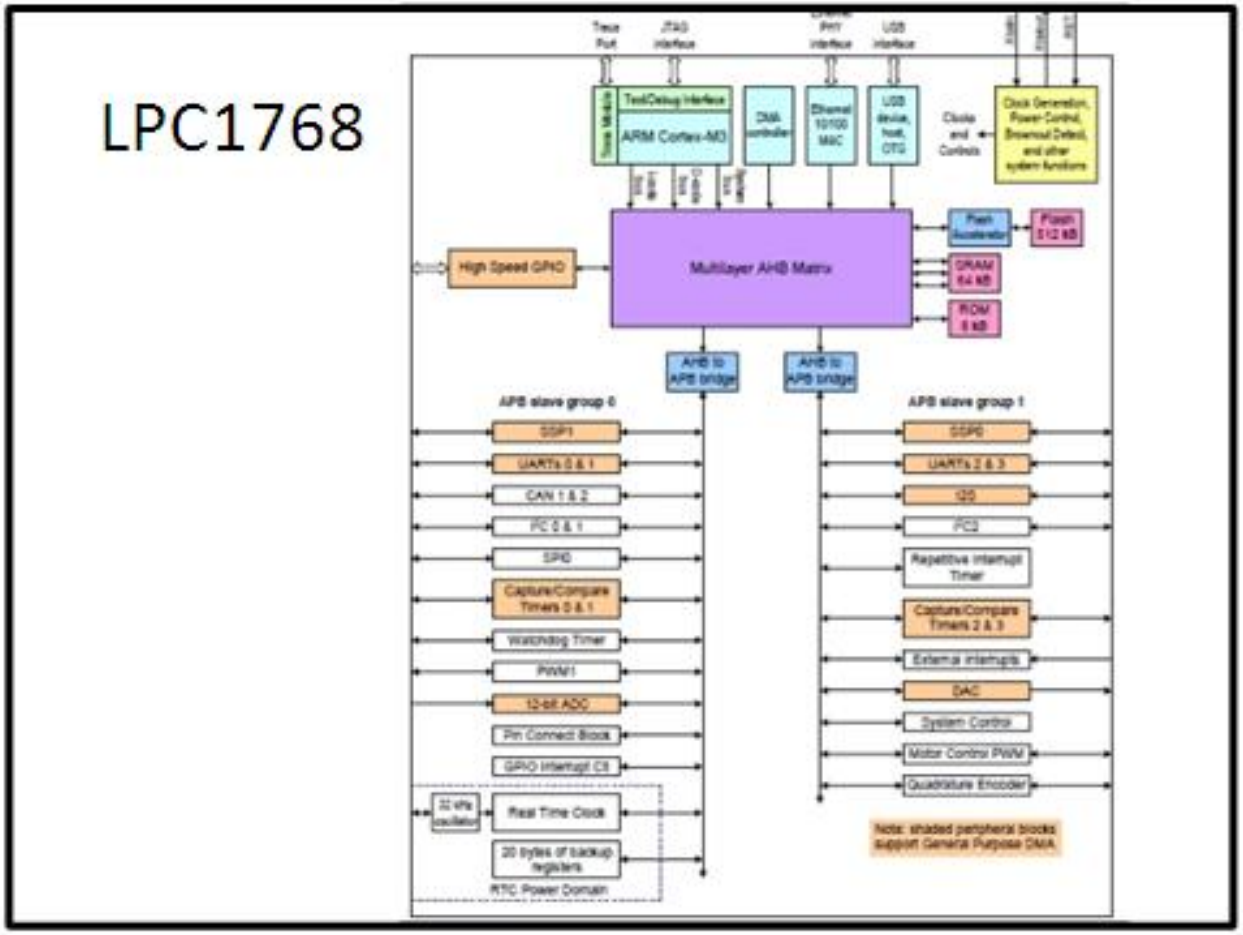
The below table shows the registers associated with LPC1768 Timer module.

Register	Description
IR	Interrupt Register: The IR can be read to identify which of 6(4-match, 2-Capture) possible interrupt sources are pending. Writing Logic-1 will clear the corresponding interrupt.
TCR	Timer Control Register: The TCR is used to control the Timer Counter functions(enable/disable/reset).
TC	Timer Counter: The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.
PR	Prescalar Register: This is used to specify the Prescalar value for incrementing the TC.
PC	Prescale Counter: The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.
MCR	Match Control Register: The MCR is used to control the reseting of TC and generating of interrupt whenever a Match occurs.
MR0-MR3	Match Registers: The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

Applications

- eMetering Lighting
- Industrial networking
- Alarm systems
- White goods
- Motor control

Architecture of LP176XX



Architecture of LP176XX Contd..

- **On-chip flash program memory**

  The LPC17xx contain up to 512 kB of on-chip flash memory. A new two-port flash accelerator maximizes performance for use with the two fast AHB-Lite buses.

- **On-chip SRAM**

  The LPC17xx contain a total of 64 kB on-chip static RAM memory. This includes the main 32 kB SRAM, accessible by the CPU and DMA controller on a higher-speed bus, and two additional 16 kB each SRAM blocks situated on a separate slave port on the AHB multilayer matrix.

- This architecture allows CPU and DMA accesses to be spread over three separate RAMs that can be accessed simultaneously.

Memory Protection Unit(MPU):

- The LPC17xx have a Memory Protection Unit (MPU) which can be used to improve the reliability of an embedded system by protecting critical data within the user application.
- The MPU allows separating processing tasks by disallowing access to each other's data, disabling access to memory regions, allowing memory regions to be defined as read-only and detecting unexpected memory accesses that could potentially break the system.
- The MPU separates the memory into distinct regions and implements protection by preventing disallowed accesses. The MPU supports up to 8 regions each of which can be divided into 8 sub regions.

UARTs:

- The LPC17xx each contain four UARTs. In addition to standard transmit and receive data lines, UART1 also provides a full modem control handshake interface and support for RS-485/9-bit mode allowing both software address detection and automatic address detection using 9-bit mode.

- The UARTs include a fractional baud rate generator. Standard baud rates such as 115200 Bd can be achieved with any crystal frequency above 2 MHz.

Features of UART:

- Maximum UART data bit rate of 6.25 Mbit/s.
- 16 B Receive and Transmit FIFOs.
- Register locations conform to 16C550 industry standard.
- Receiver FIFO trigger points at 1 B, 4 B, 8 B, and 14 B.
- Built-in fractional baud rate generator covering wide range of baud rates without a need for external crystals of particular values.
- Auto baud capabilities and FIFO control mechanism that enables software flow control implementation.

Features of UART Contd..

- UART1 equipped with standard modem interface signals. This module also provides full support for hardware flow control (auto-CTS/RTS).
- Support for RS-485/9-bit/EIA-485 mode (UART1).
- UART3 includes an IrDA mode to support infrared communication.
- All UARTs have DMA support

USB interface

- The USB controller is available as device/Host/OTG controller on parts LPC1769/68/66/65 and as device-only controller on part LPC1764
- The Universal Serial Bus (USB) is a 4-wire bus that supports communication between a host and one or more (up to 127) peripherals.
- The host controller allocates the USB bandwidth to attached devices through a token-based protocol.
- The bus supports hot plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

USB device controller

- The device controller enables 12 Mbit/s data exchange with a USB Host controller. It consists of a register interface, serial interface engine, endpoint buffer memory, and a DMA controller.
- The serial interface engine decodes the USB data stream and writes data to the appropriate endpoint buffer.
- The status of a completed USB transfer or error condition is indicated via status registers.
- An interrupt is also generated if enabled. When enabled, the DMA controller transfers data between the endpoint buffer and the on-chip SRAM.

Features of USB Interface

- Fully compliant with *USB 2.0 specification (full speed)*.
- Supports 32 physical (16 logical) endpoints with a 4 kB endpoint buffer RAM.
- Supports Control, Bulk, Interrupt and Isochronous endpoints.
- Scalable realization of endpoints at run time.

Features of USB Interface contd...

- Endpoint Maximum packet size selection (up to USB maximum specification) by software at run time.
- Supports SoftConnect and Good Link features.
- While USB is in the Suspend mode, the part can enter one of the reduced power modes and wake up on USB activity.
- Supports DMA transfers with all on-chip SRAM blocks on all non-control endpoints.
- Allows dynamic switching between CPU-controlled slave and DMA modes.
- Double buffer implementation for Bulk and Isochronous endpoints.

USB host controller

The host controller enables full- and low-speed data exchange with USB devices attached to the bus. It consists of a register interface, a serial interface engine, and a DMA controller. The register interface complies with the OHCI specification.

Features

- OHCI compliant.
- One downstream port.
- Supports port power switching.

ADC:

12 Bit ADC

The LPC17xx contain a single 12-bit successive approximation ADC with eight channels and DMA support.

Features:

- 12-bit successive approximation ADC.
- Input multiplexing among 8 pins.
- Power-down mode.
- Measurement range VREFN to VREFP.
- 12-bit conversion rate: 200 kHz.
- The ramp voltage is retained till the next pulse.

ADC cont...

- Individual channels can be selected for conversion.
- Burst conversion mode for single or multiple inputs.
- Optional conversiond on transition of input pin or Timer Match signal.
- Individual result registers for each ADC channel to reduce interrupt overhead.
- DMA support.

ADC Cont..

10-bit DAC

The DAC allows generating a variable analog output. The maximum output value of the DAC is VREFP.

Remark: The DAC is available on parts LPC1769/68/67/66/65/63.

Features

- 10-bit DAC
- Resistor string architecture
- Buffered output
- Power-down mode
- Selectable output drive
- Dedicated conversion timer
- DMA support

PWM

- The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on the LPC17xx.
- The Timer is designed to count cycles of the system derived clock and optionally switch pins, generate interrupts or perform other actions when specified timer values occur, based on seven match registers.
- The PWM function is in addition to these features, and is based on match register events.

PWM Cont...

- The ability to separately control rising and falling edge locations allows the PWM to be used for more applications.
- For instance, multi-phase motor control typically requires three non-overlapping PWM outputs with individual control of all three pulse widths and positions.
- Two match registers can be used to provide a single edge controlled PWM output.

PWM Cont..

- One match register (PWMMR0) controls the PWM cycle rate, by resetting the count upon match.
- The other match register controls the PWM edge position.
- Additional single edge controlled PWM outputs require only one match register each, since the repetition rate is the same for all PWM outputs.
- Multiple single edge controlled PWM outputs will all have a rising edge at the beginning of each PWM cycle, when an PWMMR0 match occurs.

PWM Cont..

- Three match registers can be used to provide a PWM output with both edges controlled. Again, the PWMMR0 match register controls the PWM cycle rate.
- The other match registers control the two PWM edge positions.
- Additional double edge controlled PWM outputs require only two match registers each, since the repetition rate is the same for all PWM outputs.

PWM Cont..

Features

- One PWM block with Counter or Timer operation (may use the peripheral clock or one of the capture inputs as the clock source).
- Seven match registers allow up to 6 single edge controlled or 3 double edge controlled PWM outputs, or a mix of both types. The match registers also allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.

COMPARE PAM PWM PPM

PAM	PWM	PPM
Amplitude is varied	Width is varied	Position is varied
Bandwidth depends on the width of the pulse	Bandwidth depends on the rise time of the pulse	Bandwidth depends on the rise time of the pulse
Instantaneous transmitter power varies with the amplitude of the pulses	Instantaneous transmitter power varies with the amplitude and width of the pulses	Instantaneous transmitter power remains constant with the width of the pulses
System complexity is high	System complexity is low	System complexity is low
Noise interference is high	Noise interference is low	Noise interference is low
It is similar to amplitude modulation	It is similar to frequency modulation	It is similar to phase modulation

Motor control PWM

•The motor control PWM is a specialized PWM supporting 3-phase motors and other combinations.
•Feedback inputs are provided to automatically sense rotor position and use that information to ramp speed up or down.
• An abort input is also provided that causes the PWM to immediately release all motor drive outputs.
•At the same time, the motor control PWM is highly configurable for other generalized timing, counting, capture, and compare applications.

RTC (Real Time Clock)

- The RTC is a set of counters for measuring time when system power is on, and optionally when it is off.
- The RTC on the LPC17xx is designed to have extremely low power consumption, i.e. less than 1 mA.
- The RTC will typically run from the main chip power supply, conserving battery power while the rest of the device is powered up.
- When operating from a battery, the RTC will continue working down to 2.1 V. Battery power can be provided from a standard 3 V Lithium button cell.

RTC

- An ultra-low power 32 kHz oscillator will provide a 1 Hz clock to the time counting portion of the RTC, moving most of the power consumption out of the time counting function.
- The RTC includes a calibration mechanism to allow fine-tuning the count rate in a way that will provide less than 1 second per day error when operated at a constant voltage and temperature.
- The RTC contains a small set of backup registers (20 bytes) for holding data while the main part of the LPC17xx is powered off.
- The RTC includes an alarm function that can wake up the LPC17xx from all reduced power modes with a time resolution of 1 s.

RTC Cont..

Features

- Measures the passage of time to maintain a calendar and clock.
- Ultra low power design to support battery powered systems.
- Provides Seconds, Minutes, Hours, Day of Month, Month, Year, Day of Week, and Day of Year.
- Dedicated power supply pin can be connected to a battery or to the main 3.3 V.
- Periodic interrupts can be generated from increments of any field of the time registers.
- Backup registers (20 bytes) powered by VBAT.
- RTC power supply is isolated from the rest of the chip.

WDT(Watchdog Timer)

- For those embedded systems that can't be constantly watched by a human, watchdog timers may be the solution Most embedded systems need to be self- reliant
- It's not usually possible to wait for someone to reboot them if the software hangs.
- Some embedded designs, such as space probes, are simply not accessible to human operators
- If their software ever hangs, such systems are permanently disabled
- In other cases, the speed with which a human operator might reset the system would be too slow to meet the uptime requirements of the product

WDT Cont..

- A watchdog timer is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur
- Generally speaking, a watchdog timer is based on a counter that counts down from some initial value to zero
- The embedded software selects the counter's initial value and periodically restarts it
- If the counter ever reaches zero before the software restarts it, the software is presumed to be malfunctioning and the processor's reset signal is asserted
- The processor (and the embedded software it's running) will be restarted as if a human operator had cycled the power

WDT Cont..

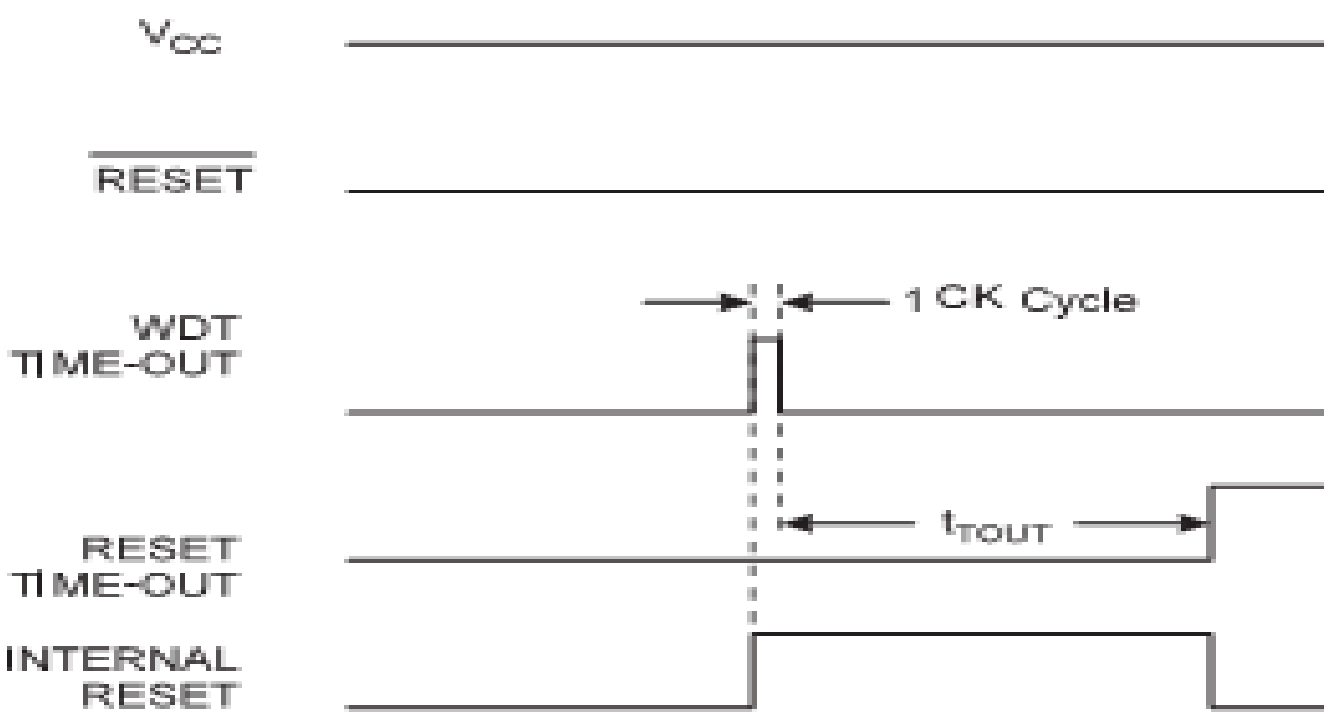
The purpose of the watchdog is to reset the microcontroller within a reasonable amount of time if it enters an erroneous state. When enabled, the watchdog will generate a system reset if the user program fails to ‘feed’ (or reload) the watchdog within a predetermined amount of time.

- Watchdog timer is a chip external to the processor.
- Could also be included within the same chip as the CPU- many microcontrollers.

WDT setup

Figure 1: A typical watchdog setup

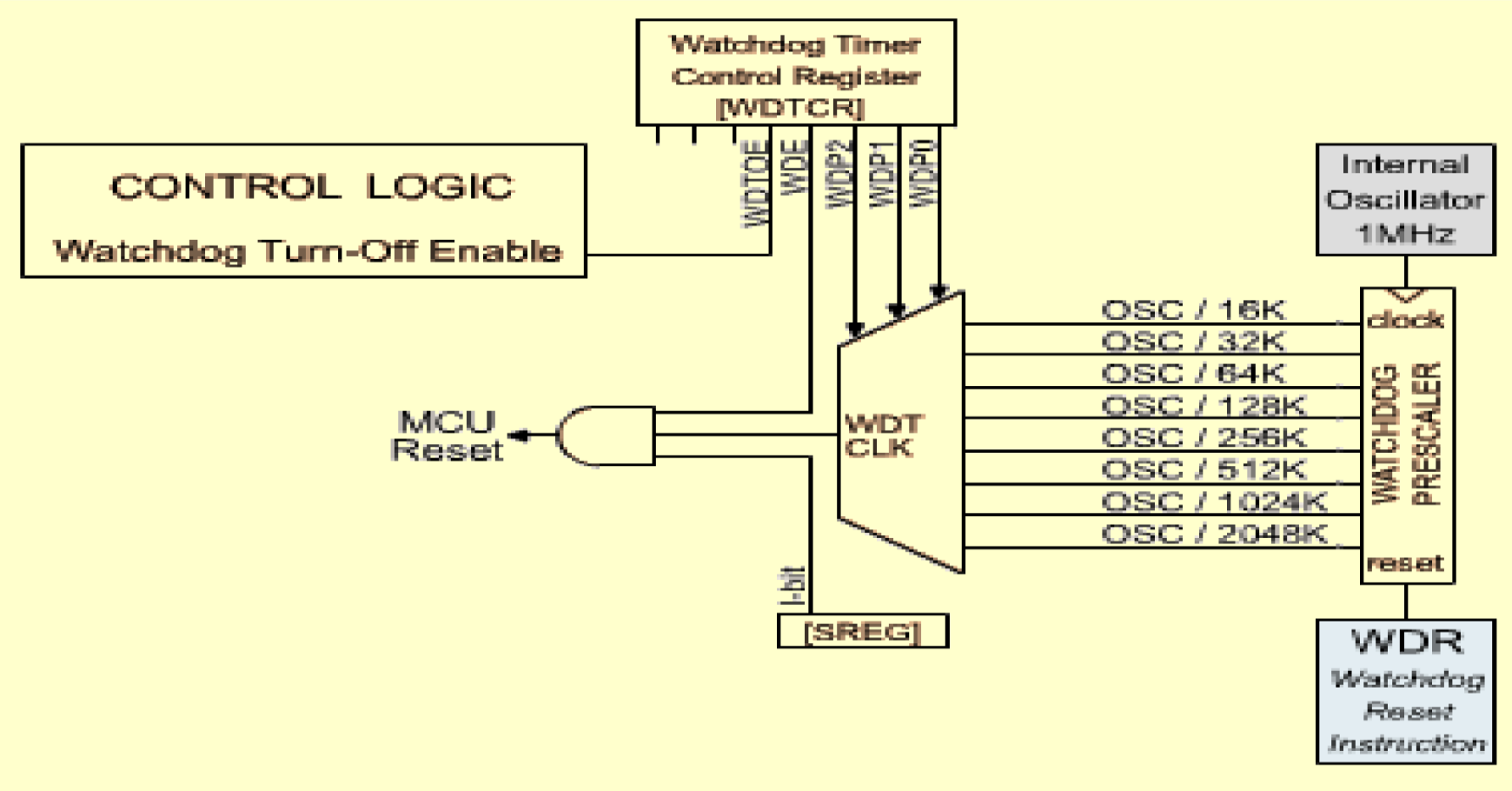
WDT cont..


V_{CC}
RESET
1 CK Cycle
WDT TIME-OUT
t_{TOUT}
RESET TIME-OUT
INTERNAL RESET

WDT cont..

- After watchdog timer counts up to maximum, it generates a short pulse duration 1 clock cycle. This pulse triggers internal reset timer counting up to tout
- AVR watchdog timer is distinct clock generator which runs at 1 MHz
- Watchdog timer has a prescaler module. So reset interval can be selected by adjusting the prescaler.
- Generally there are three things to do while controlling watchdog timer: enable, disable, and set prescaler.

WDT cont…


Watchdog Timer
Control Register
[WDTCR]
CONTROL LOGIC
Watchdog Turn-Off Enable
WDTOE
WDE
WDP2
WDP1
WDP0
Internal
Oscillator
1MHz
OSC / 16K
OSC / 32K
OSC / 64K
OSC / 128K
OSC / 256K
OSC / 512K
OSC / 1024K
OSC / 2048K
clock
WATCHDOG
PRESCALER
reset
MCU
Reset
WDT
CLK
I-bit
[SREG]
WDR
Watchdog
Reset
Instruction

WDTCR Register

Bit 4 – WDTOE: Watchdog Turn-off Enable

Bit 3 – WDE: Watchdog Enable

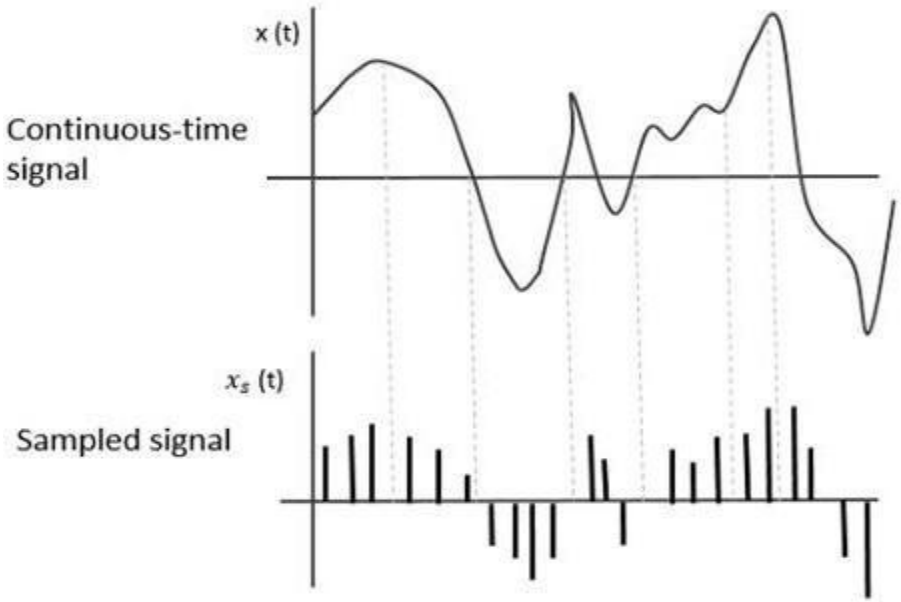
Bits 2..0 – WDP2, WDP1, WDP0: Watchdog Timer Prescaler 2, 1, and 0

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

WDTCR Register Cont..

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

SAMPLING



❖The **sampling period** is the time difference between two consecutive samples, It is the inverse of the sampling frequency

$$Sampling\ Frequency = \frac{1}{T_s} = f_s$$

Where,

- T_s is the sampling time
- f_s is the sampling frequency or the sampling rate

SAMPLING Cont…

Nyquist Rate: $f_S = 2W$

Where,

- f_S is the sampling rate
- **W** is the highest frequency

This rate of sampling is called as **Nyquist rate**.

Sampling Theorem:

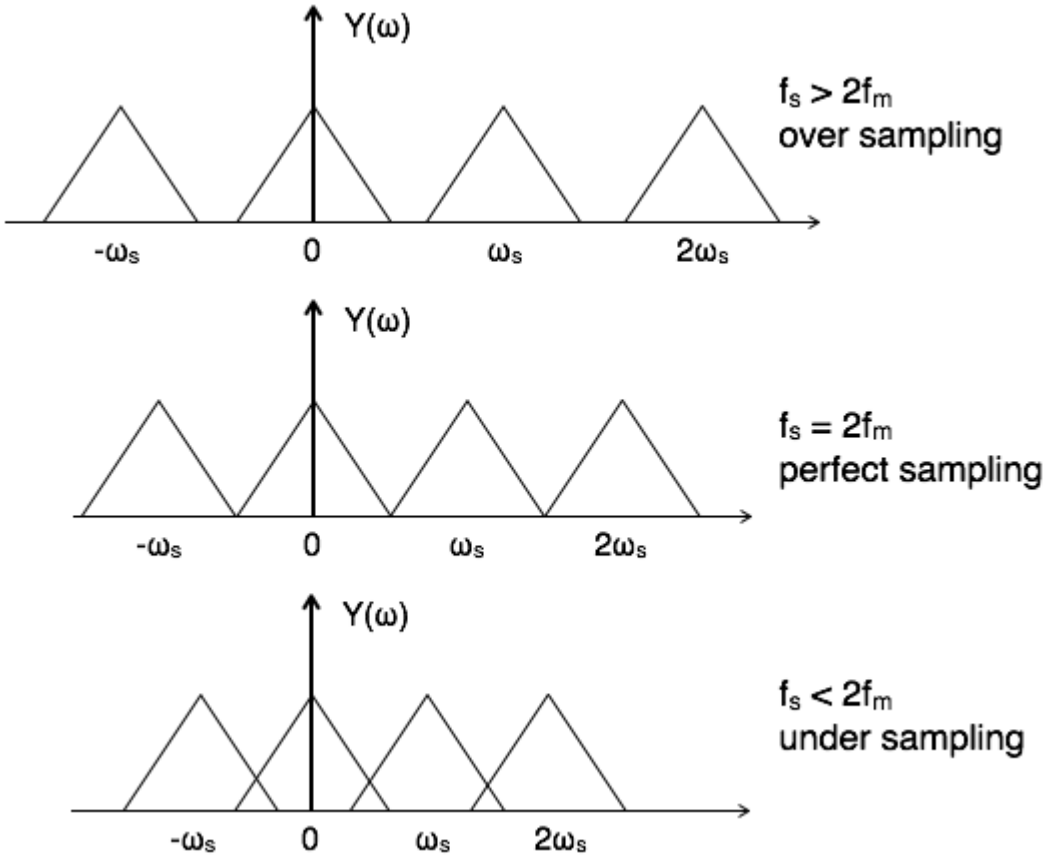
statement: A continuous time signal can be represented in its samples and can be recovered back when sampling frequency f_s is greater than or equal to the twice the highest frequency component of message signal. i. e.

$$f_s \geq 2f_m.$$

Aliasing effect: Aliasing is the phenomenon in which higher frequency components are combined with lower frequency components in spectrum of its sampled version. $f_s < 2f_m$

- By using anti-aliasing filters.

SAMPLING Cont…



SAMPLING Cont…

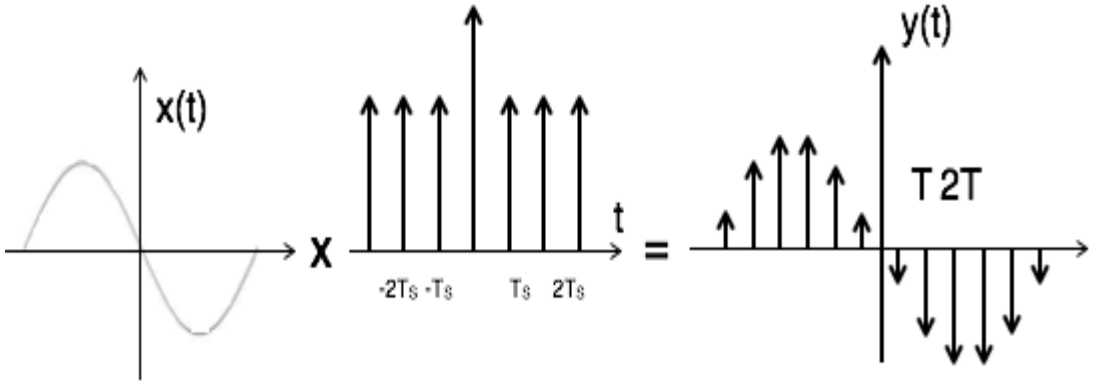
TYPES OF SAMPLING:

There are 3 sampling techniques,They are:

1.Ideal sampling or impulse sampling or instantaneous sampling:

- The instantaneous sampling has a train of impulses.
- The pulse width of the samples has almost zero value.
- It is the product of message signal m(t) with a unit impulse train δ(t) gives sampled signal.

i.e., $m_s(t)=m(t)\delta(t)$



PROGRAMMABLE DSP (P-DSP) PROCESSORS

DSP Applications

DSP applications are often real time but with a wide variety of sample rates

- High rates
  - Radar
  - Video
- Medium rates
  - Audio
  - Speech
- Low rates
  - Weather
  - Finance

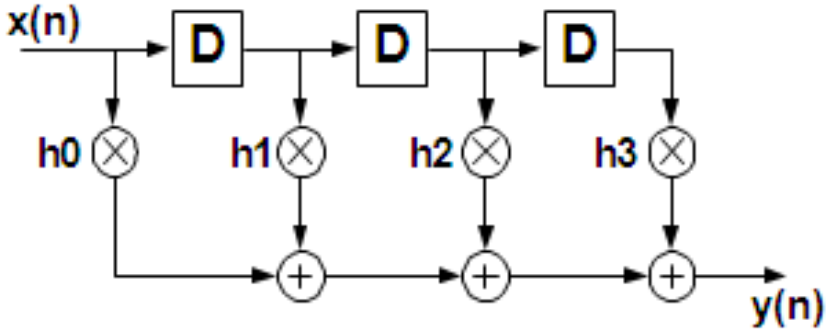
With different demands on

- numeric representation
  - float or fixed
  - and nmber of bits
- Throughput/speed
- Power/energy dissipation
- Cost

DSP features

Fast Multiply/Accumulate (MAC)

- FIR
- FFT
- etc.



- Multiple Access Memories
- Specialized addressing modes
- Specialized execution control (loops)
- Specialized interfaces, e.g. AD/DA

Standard DSP Alternatives

PCs or Workstations

- Non-real time
- low requirements

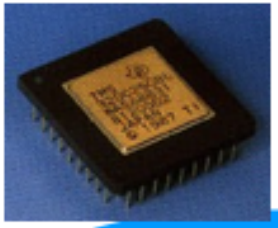
General purpose microprocessors

- slower for DSP applications
- might be one µ proc. there anyway

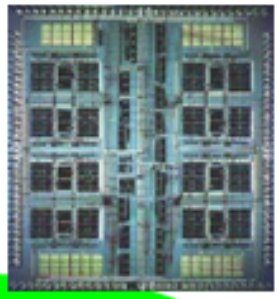
Custom

- performance
- low cost at volume
- High development cost

Standard Processors vs. Special Purpose



Algorithm



Standard Processor

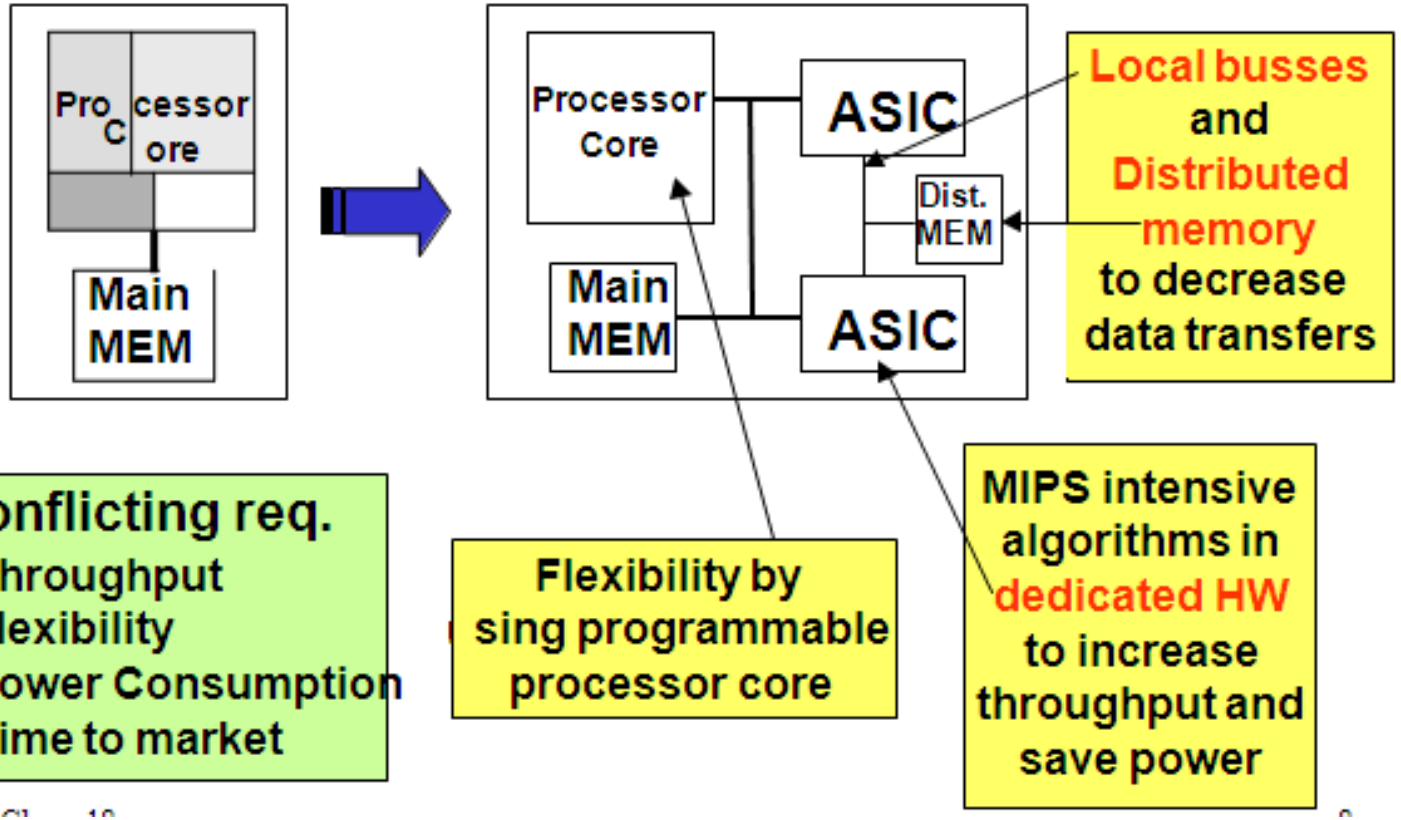
- Programmable
- Low Design cost
- Standard Interface
- Good supply of tools

Processor Cores
Domain Specific Processors
etc.

Special Purpose

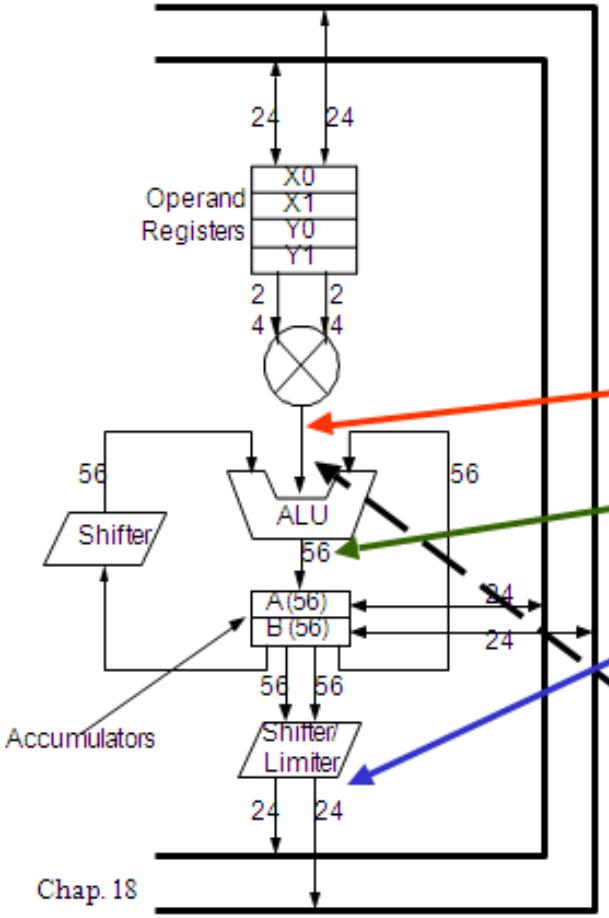
- High Calculation Capacity
- Low Power
- User defined Interface
- Variable Wordlength
- Low Price at Volume

Architectural Partitioning



Fixed point DSP

Motorola DSP56000x



- Usually DSP has single cycle multiplier, may be pipelined
- Double wordlength out
- + guard bits
- scaling
- Altenative is mult with reduced wordlength output, e.g. 24

9

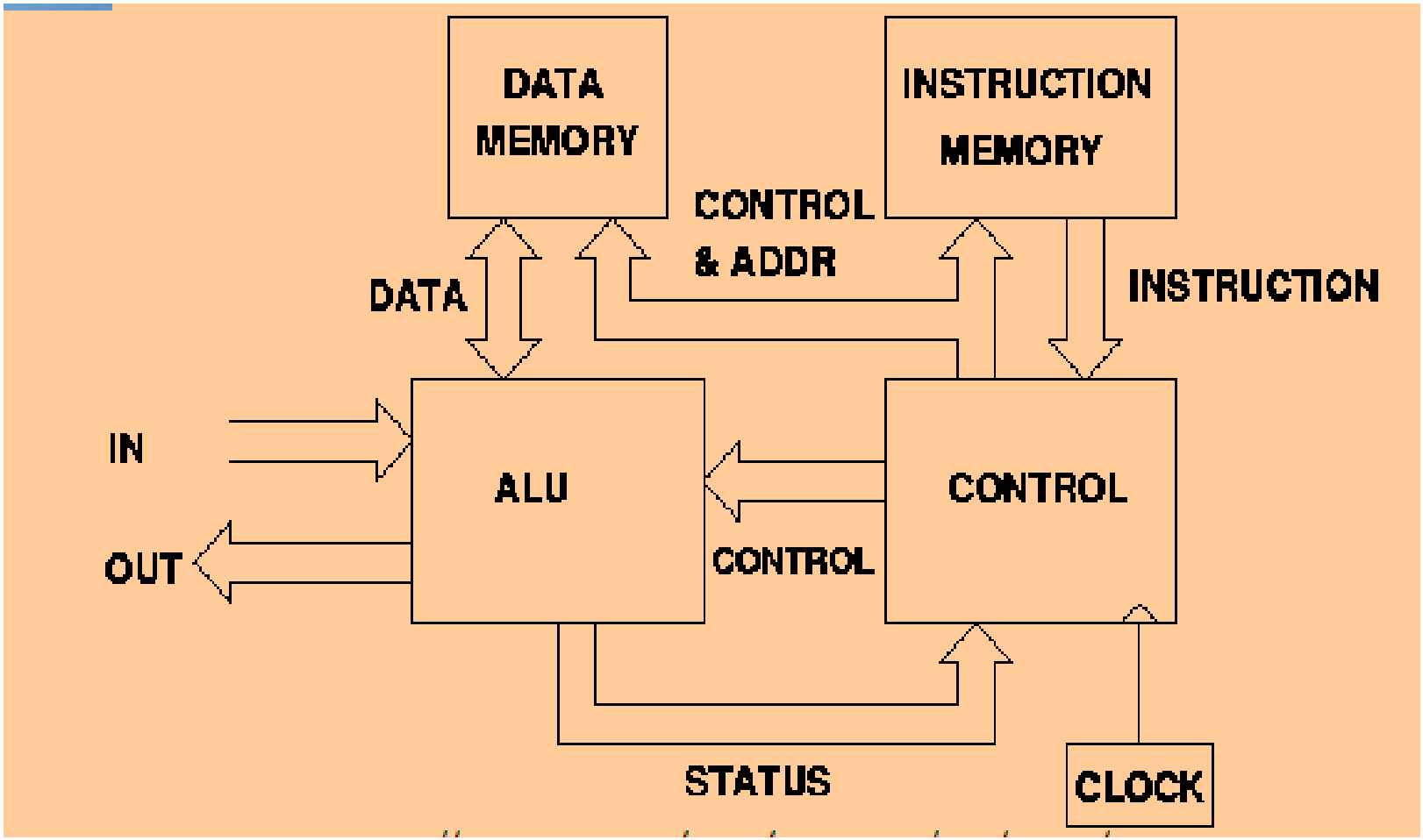
How to classify processors

- Categorized by memory organization
  - Von-Neumann architecture
  - Harvard architecture

Harvard architecture

- Separate memory into 2 types
  - Program memory
  - Data memory
- Used in MCS-51, MIPS etc.

Harvard Architecture Cont..

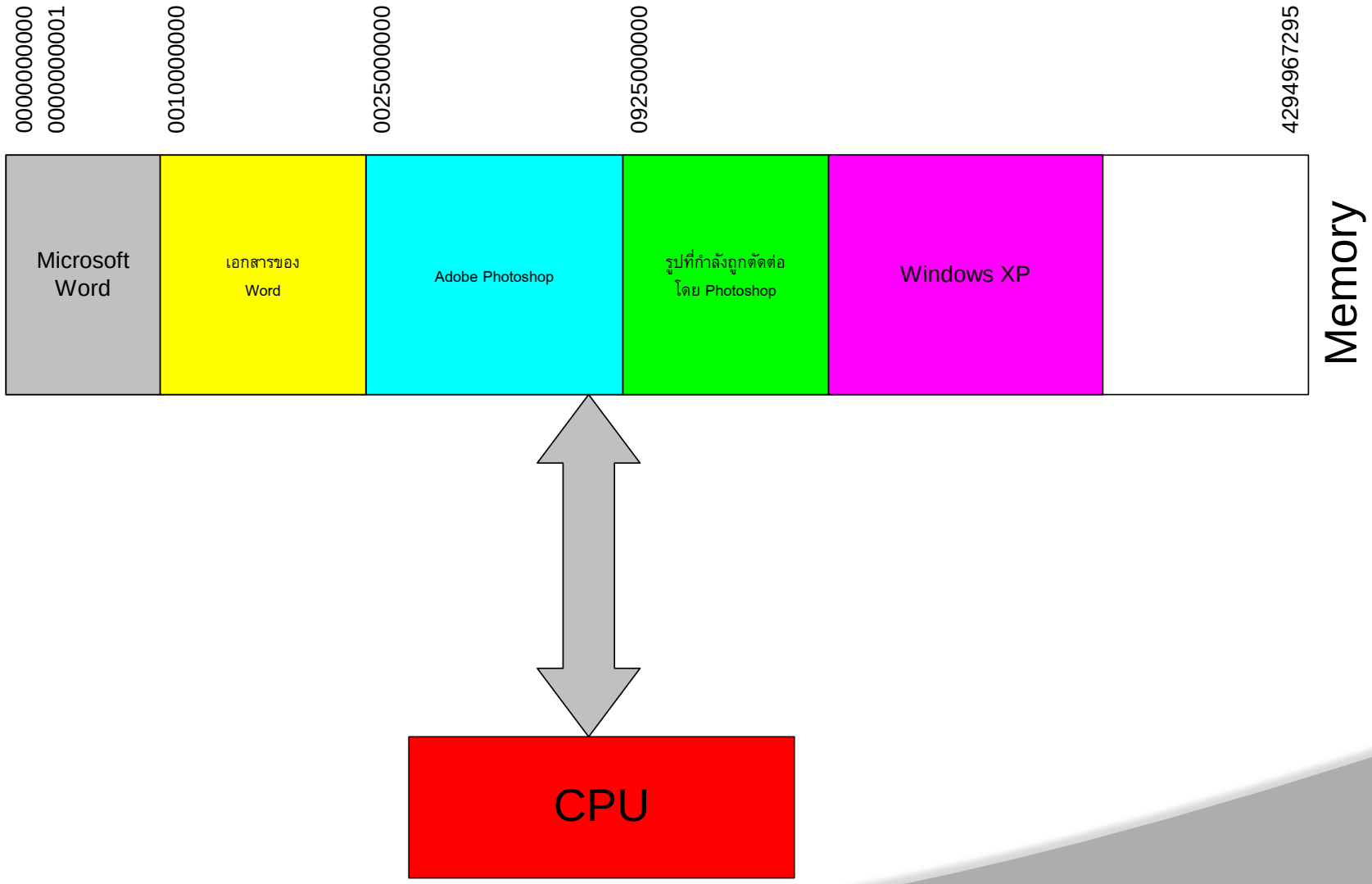


Von-Neumann architecture

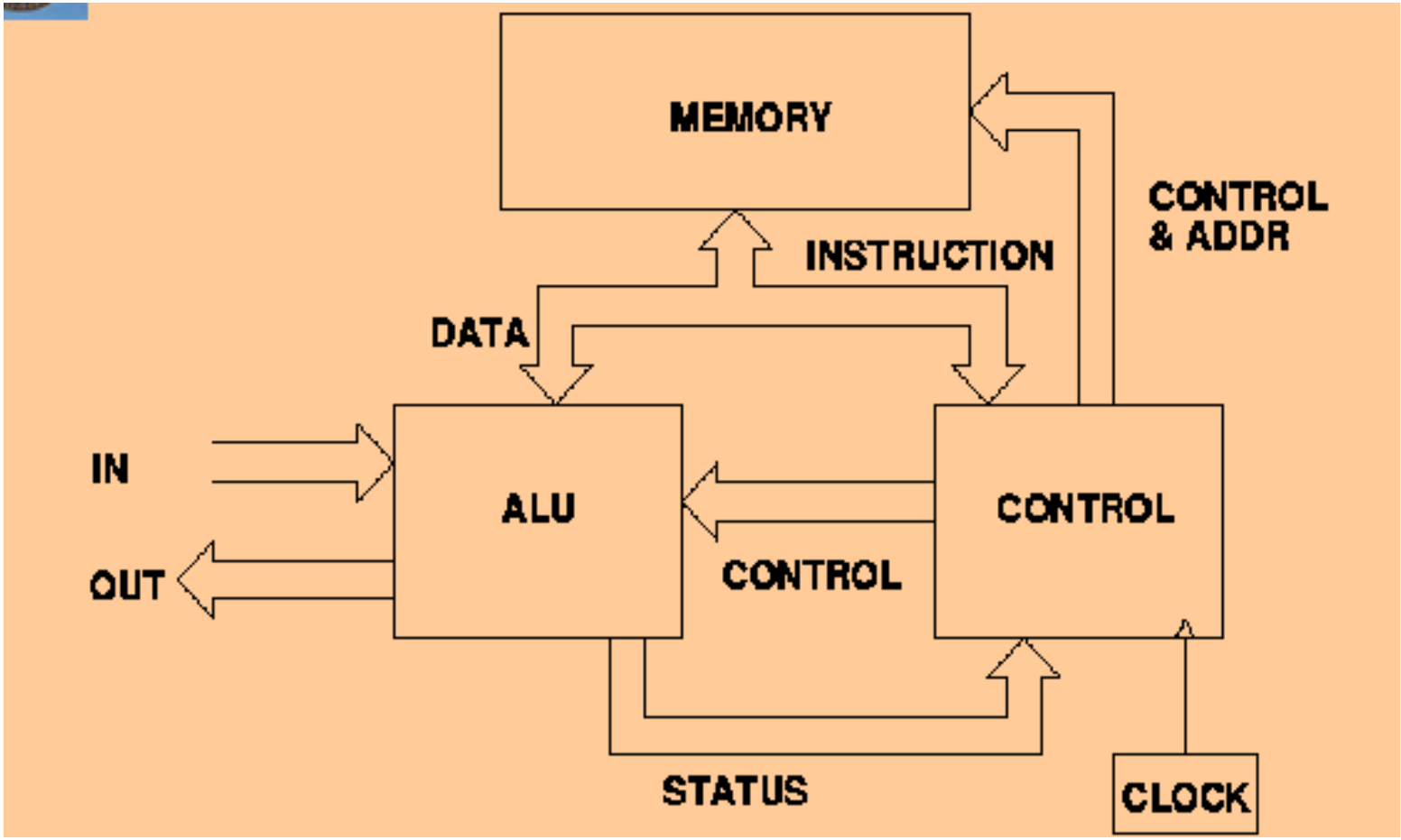
- Combine program and data in 1 chunk of memory

Example : 80x86 architecture

Von-Neumann architecture cont..

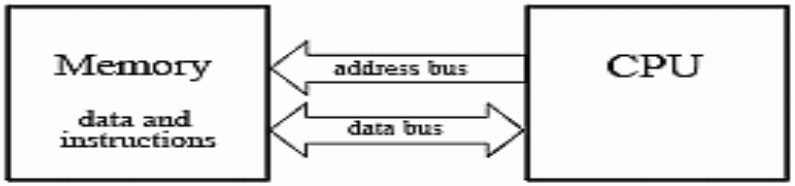


Von-Neumann architecture cont..

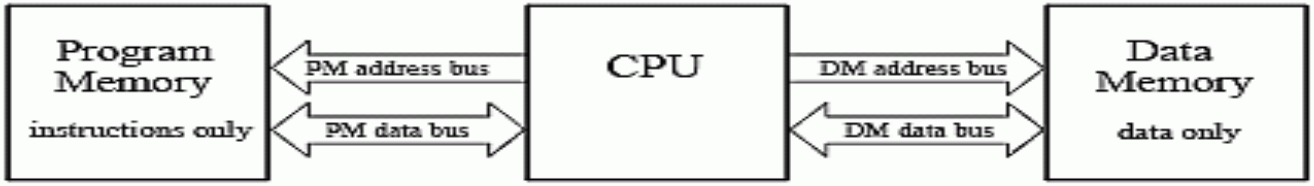


DIFFERENCE B/W Von-Neumann AND Harvard architecture:

a. Von Neumann Architecture (*single memory*)



b. Harvard Architecture (*dual memory*)



c. Super Harvard Architecture (*dual memory, instruction cache, I/O controller*)

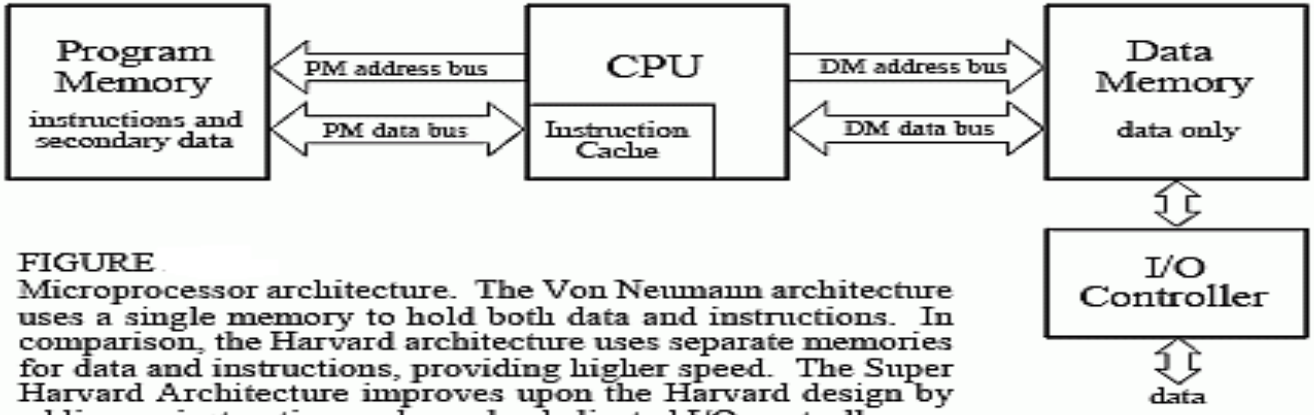


FIGURE
Microprocessor architecture. The Von Neumann architecture uses a single memory to hold both data and instructions. In comparison, the Harvard architecture uses separate memories for data and instructions, providing higher speed. The Super Harvard Architecture improves upon the Harvard design by adding an instruction cache and a dedicated I/O controller.

CISC FEATURES

a. Complex instruction set computer
b. Large number of instructions (~200-300 instructions)
c. Specialized complex instructions
d. Many different addressing modes
e. Variable length instruction format
f. Memory to memory instruction
g. For Example : 68000, 80x86

RISC FEATURES

- Reduced instruction set computer
- Relatively few number of instructions (~50)
- Basic instructions
- Relatively few different addressing modes
- Fixed length instruction format
- Only load/store instructions can access memory
- Large number of registers
- Hardwired rather than micro-program control
- For Example : MIPS, Alpha, ARM etc.

CISC Vs RISC

CISC -- High Code Density

- Fewer instructions needed to specify the algorithm

RISC -- Simpler to Design & Faster to Silicon

- Higher Performance -- smaller die size
- Lower power consumption
- Easier to develop compilers to take advantage of all features

Example of CPU Architectures

- Intel: 80x86
- Motorola: 680x0
- Sun : Sparc
- Silicon Graphics : MIPS
- HP : PA-RISC
- IBM: Power PC
- Compaq: Alpha

Architecture of P-DSP

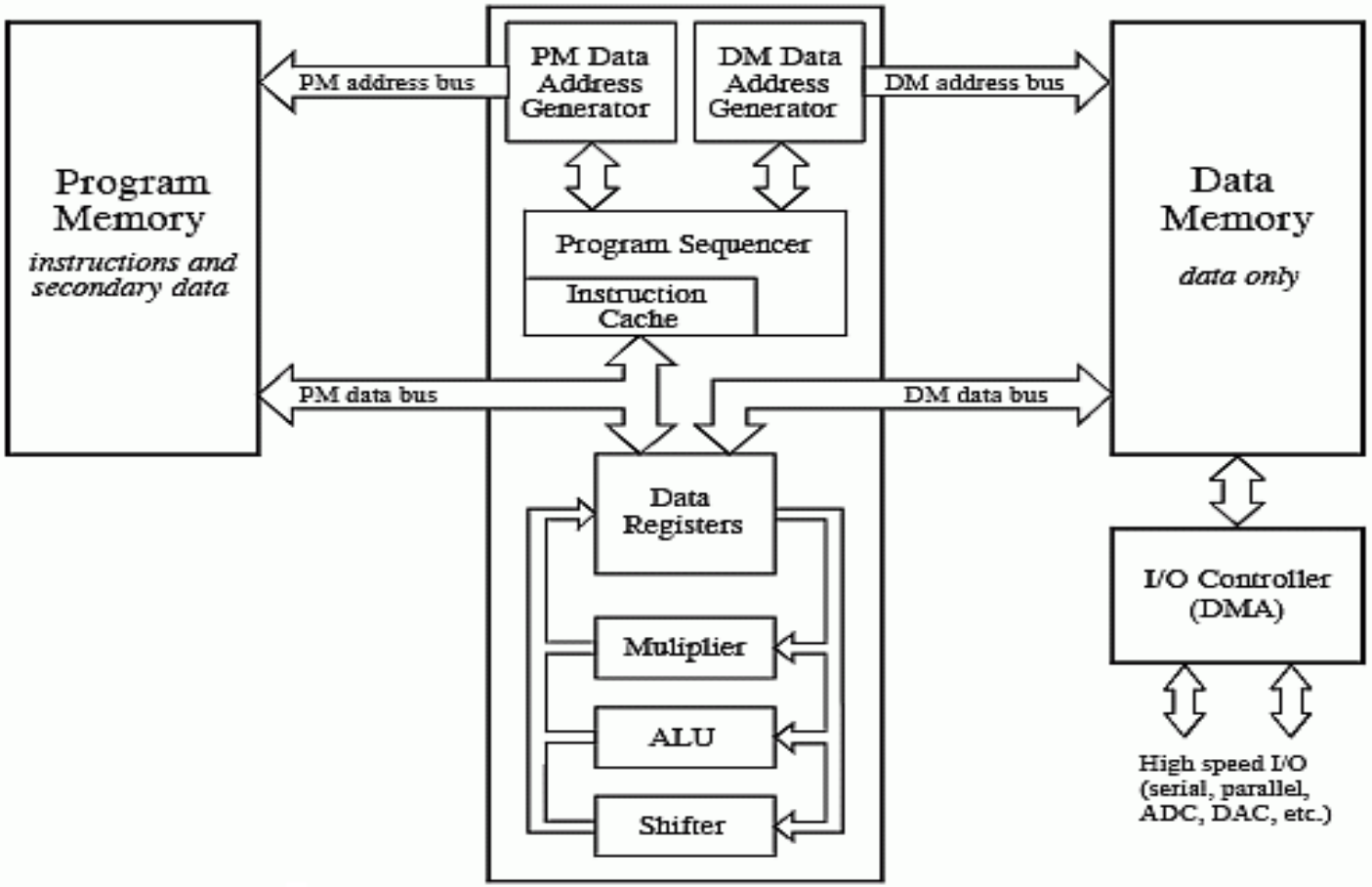


FIGURE
Typical DSP architecture. Digital Signal Processors are designed to implement tasks in parallel. This simplified diagram is of the Analog Devices SHARC DSP. Compare this architecture with the tasks needed to implement an FIR filter, as listed in Table 28-1. All of the steps within the loop can be executed in a single clock cycle.

MAC UNIT

Multiplier and Multiplier Accumulator

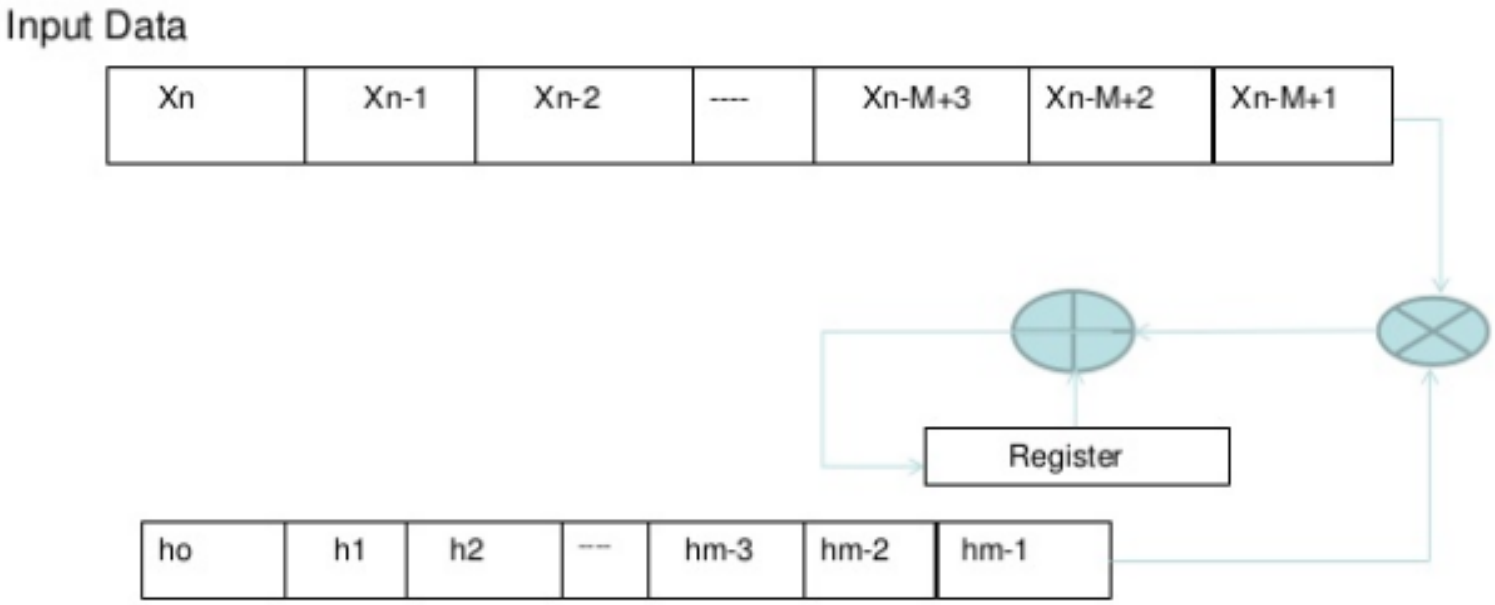
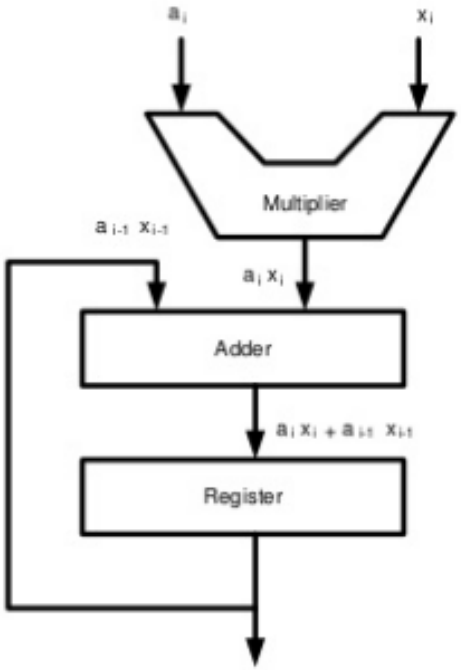


Fig. Implementation of Convolver with single multiplier/Adder

MAC UNIT Cont..

Single-Cycle MAC unit



$$\sum_{i=0}^{n} (a_i x_i)$$

Can compute a sum of *n*-products in *n* cycles

MAC in Van Neumann Architecture

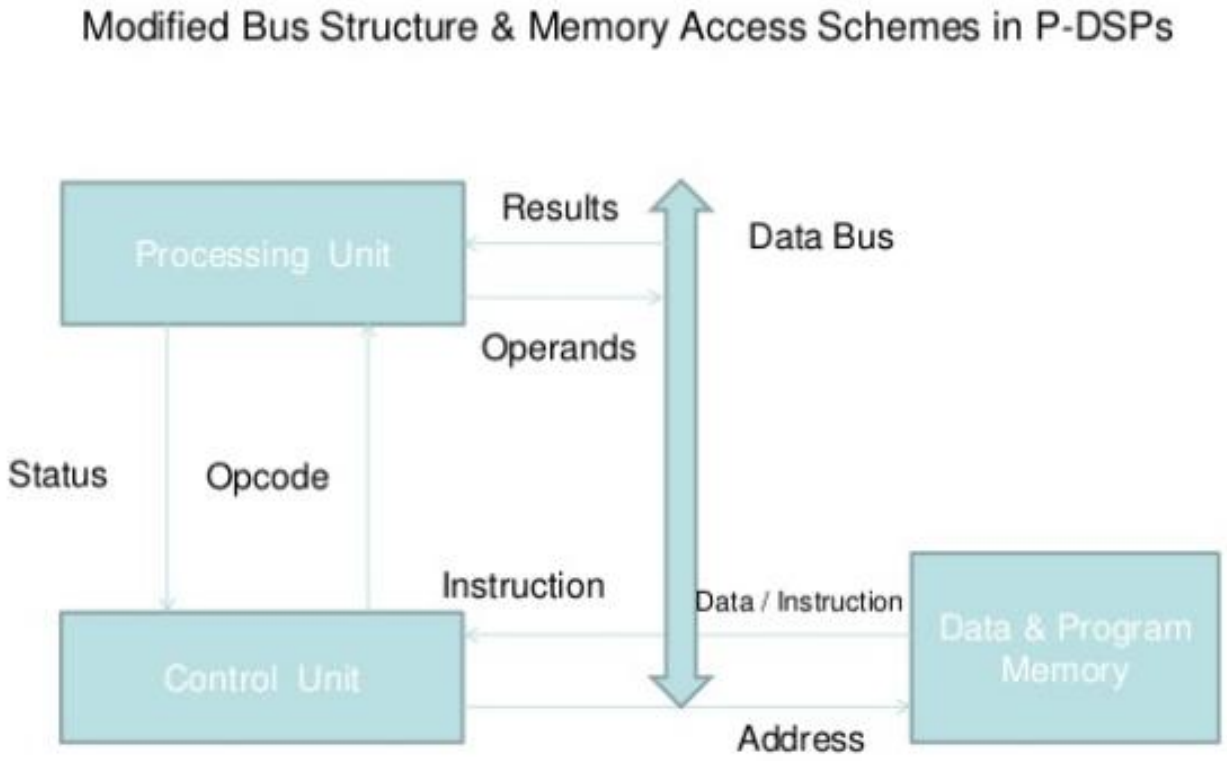


Fig: Van Neumann Architecture

MAC UNIT Cont...

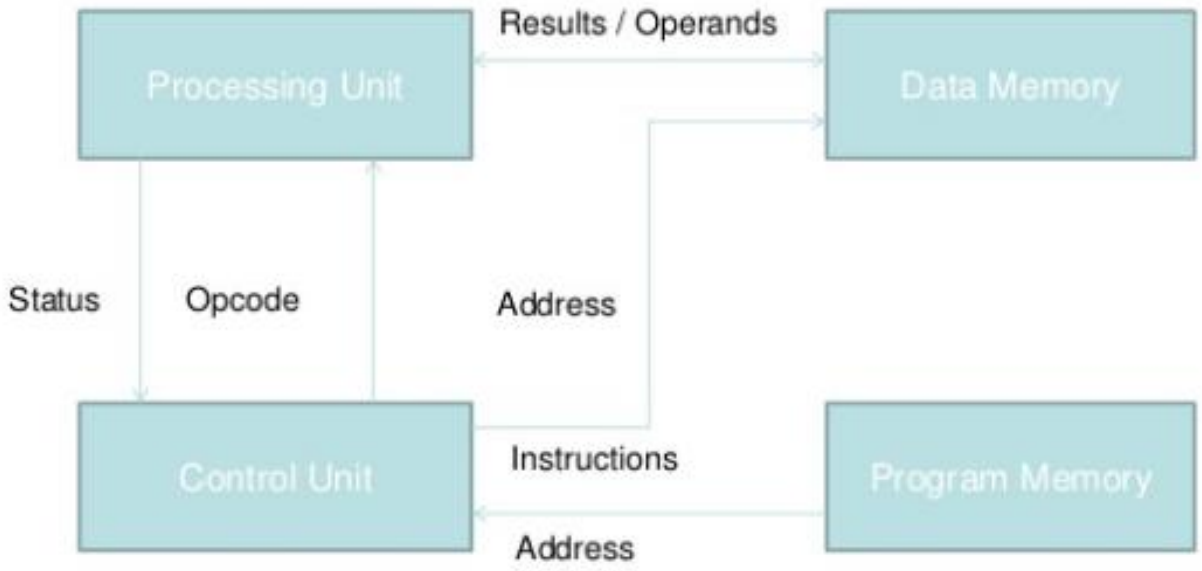


Fig: Harvard Architecture

MAC UNIT Cont...

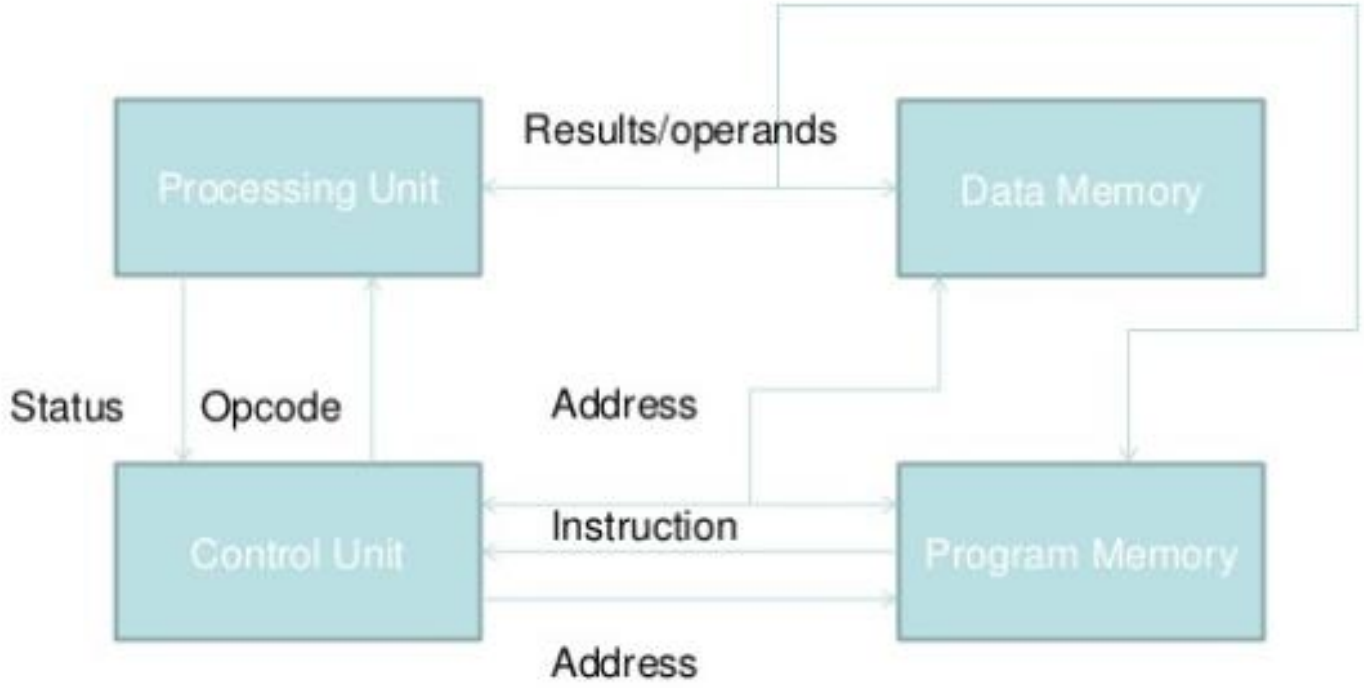


Fig: Modified harvard architecture

SHIFTERS

Shifters are used to either scale down or scale up operands or the results. The following scenarios give the necessity of a shifter

a. While performing the addition of N numbers each of n bits long, the sum can grow up to n+log2 N bits long. If the accumulator is of n bits long, then an overflow error will occur.
b. This can be overcome by using a shifter to scale down the operand by an amount of log2N.
c. Similarly while calculating the product of two n bit numbers, the product can grow up to 2n bits long.
d. Generally the lower n bits get neglected and the sign bit is shifted to save the sign of the product.

SHIFTERS CONT..

e. Finally in case of addition of two floating-point numbers, one of the operands has to be shifted appropriately to make the exponents of two numbers equal.

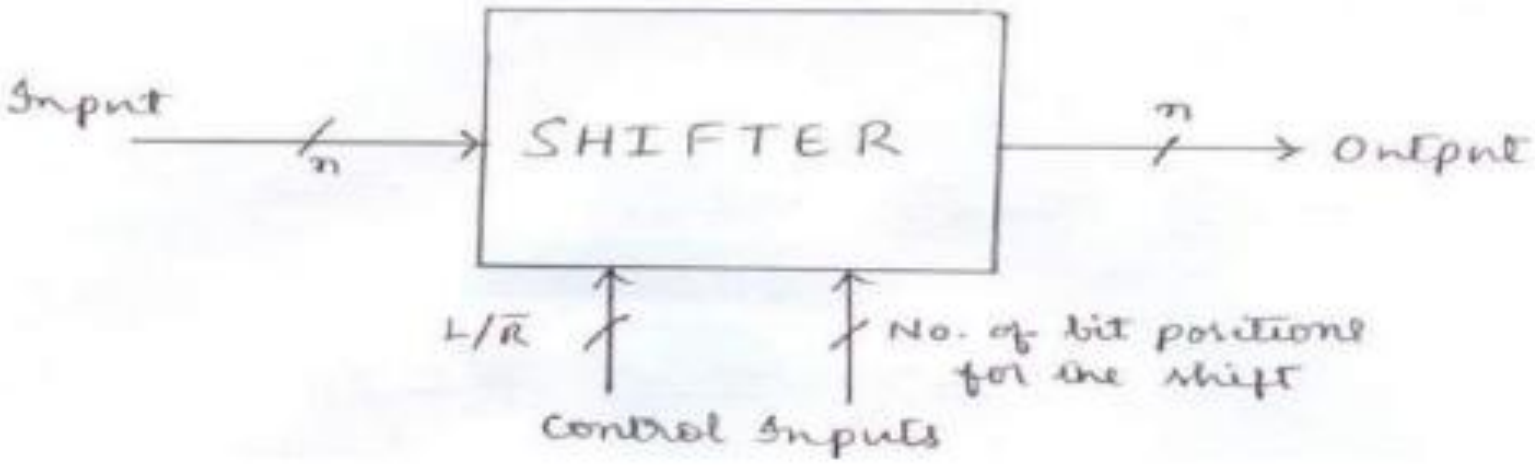
From the above cases it is clear that, a shifter is required in the architecture of a DSP

BARREL SHITERS

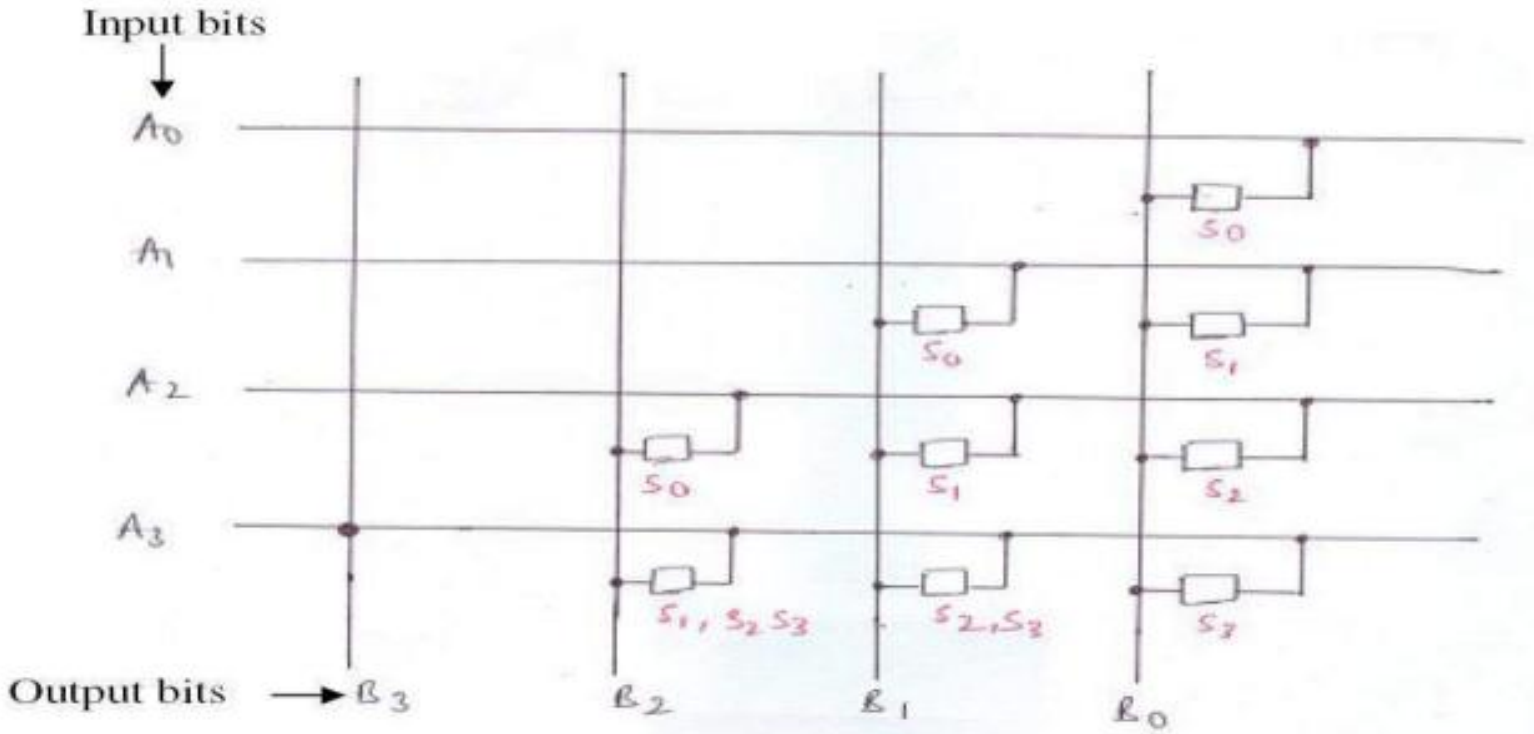
a. In conventional microprocessors, normal shift registers are used for shift operation. As it requires one clock cycle for each shift, it is not desirable for DSP applications, which generally involves more shifts.
b. In other words, for DSP applications as speed is the crucial issue, several shifts are to be accomplished in a single execution cycle. This can be accomplished using a barrel shifter, which connects the input lines representing a word to a group of output lines with the required shifts determined by its control inputs.
c. For an input of length n, log2 n control lines are required. And an Additional control line is required to indicate the direction of the shift

BARREL SHITERS Cont..

The block diagram of a typical barrel shifter is as shown in figure



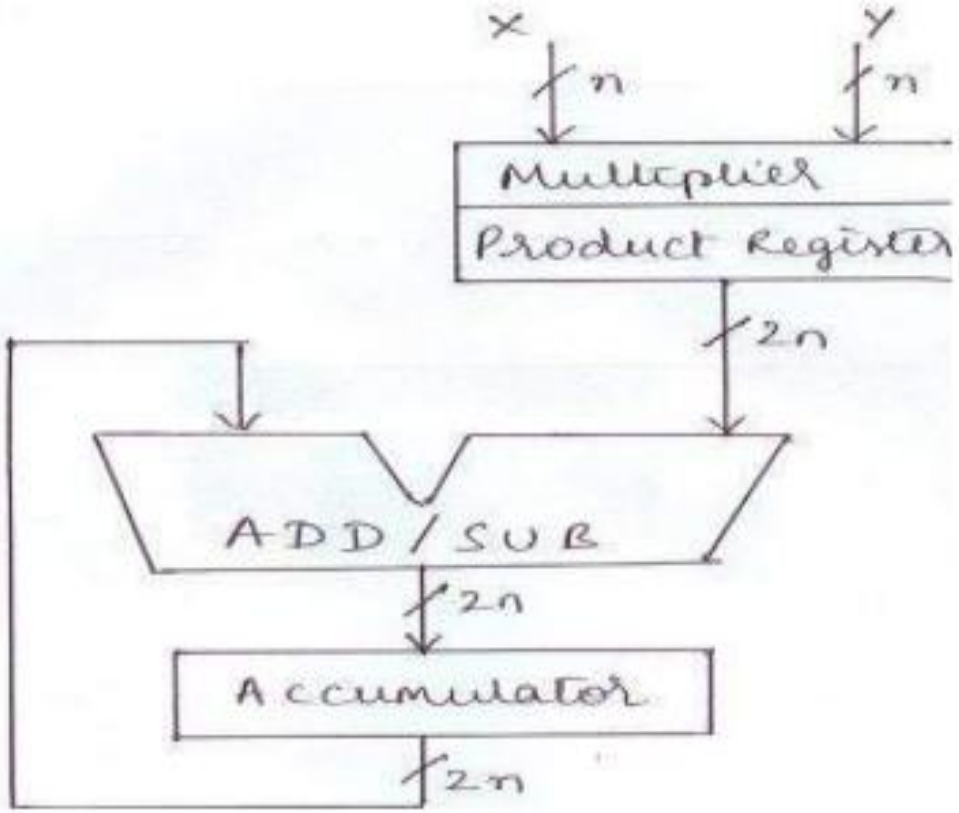
BARREL SHITERS Cont..



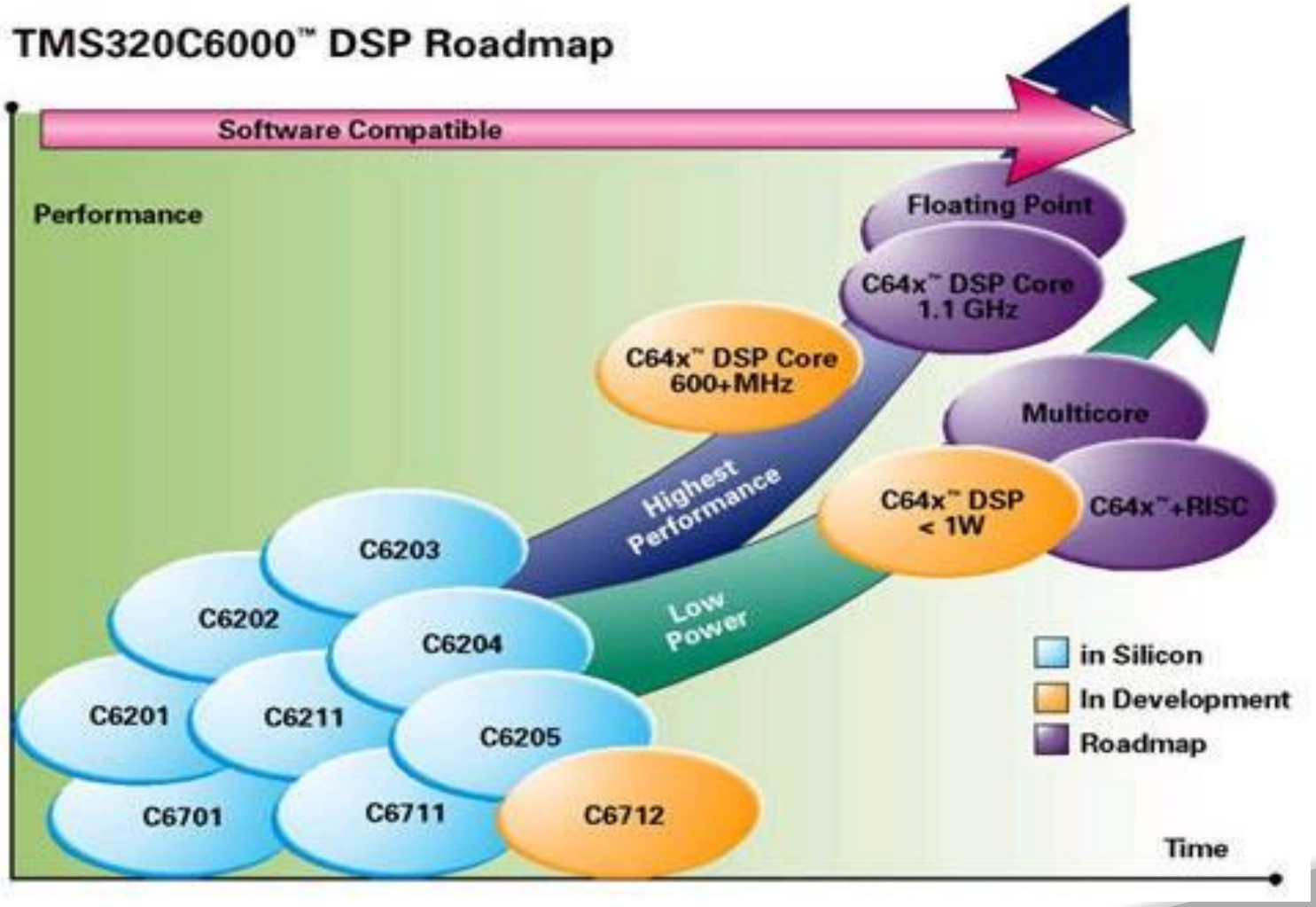
INPUT				SHIFT (SWITCH)	OUTPUT (B_3 B_2 B_1 B_0)
A_3	A_2	A_1	A_0	0 (S_0)	A_3 A_2 A_1 A_0
A_3	A_2	A_1	A_0	1 (S_1)	A_3 A_3 A_2 A_1
A_3	A_2	A_1	A_0	2 (S_2)	A_3 A_3 A_3 A_2
A_3	A_2	A_1	A_0	3 (S_3)	A_3 A_3 A_3 A_3

Fig Implementation of a 4 bit Shift Right Barrel Shifter

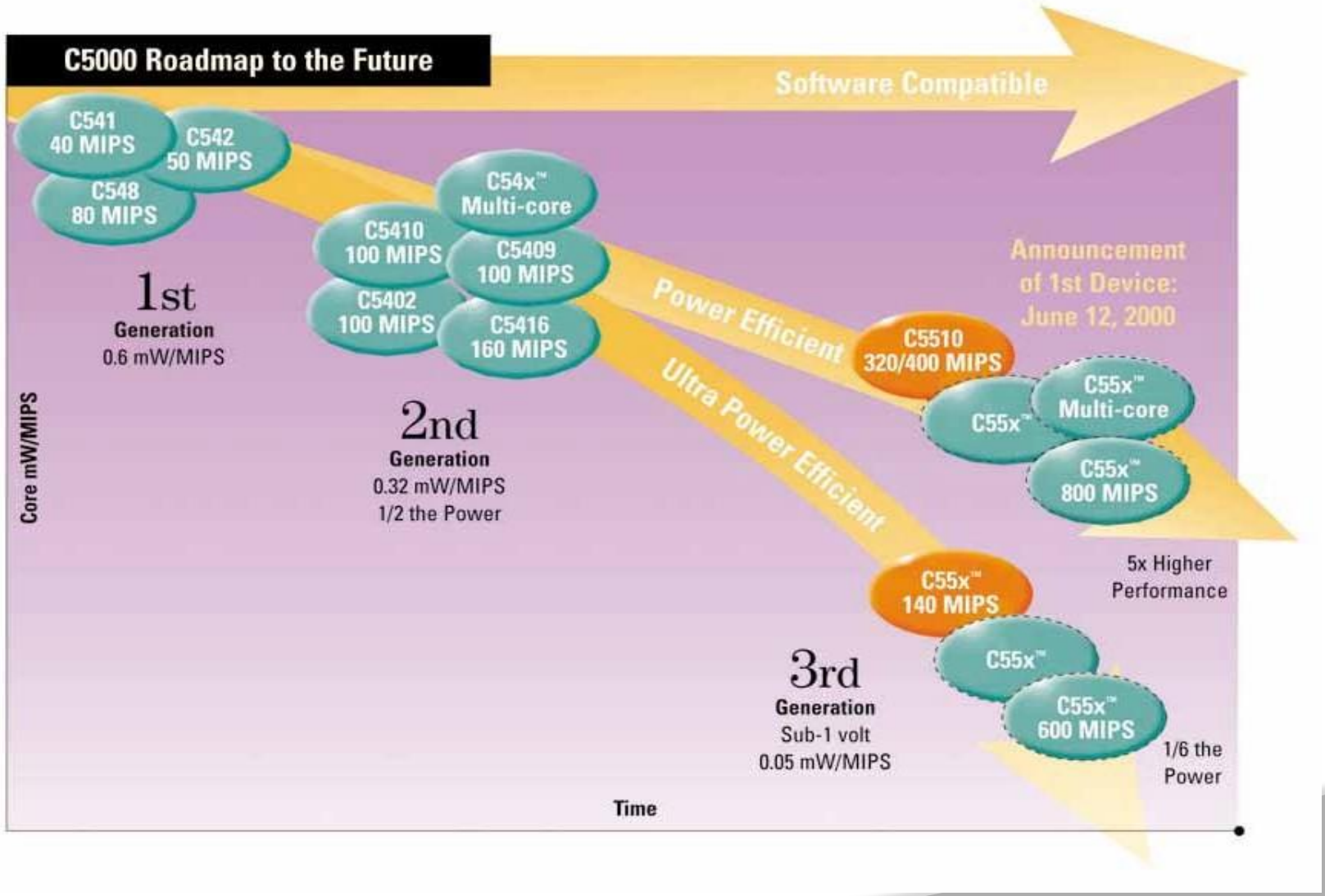
MAC UNIT


X
Y
n
n
Multiplier
Product Register
2n
ADD / SUB
2n
Accumulator
2n

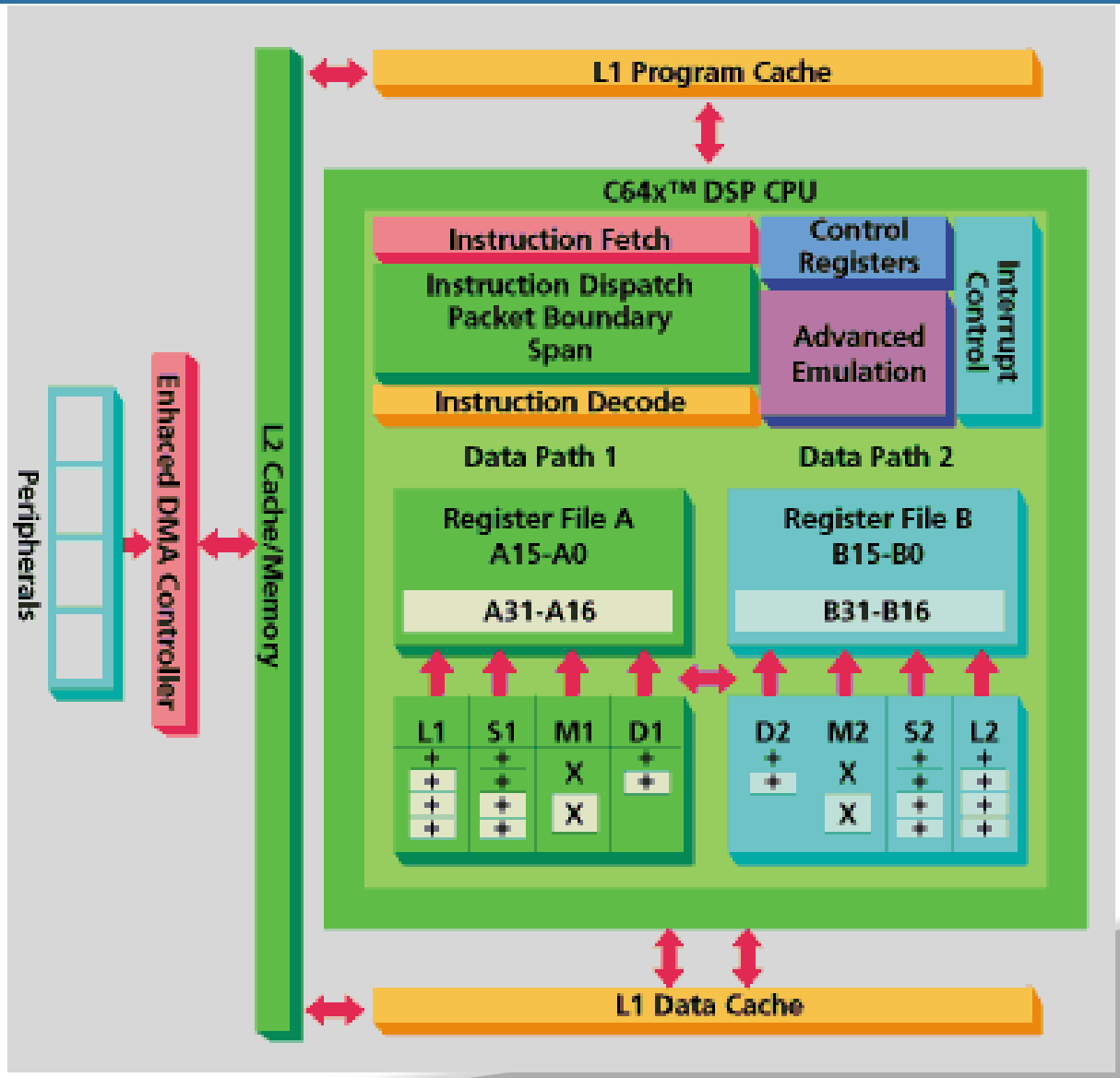
TI Processors, high speed



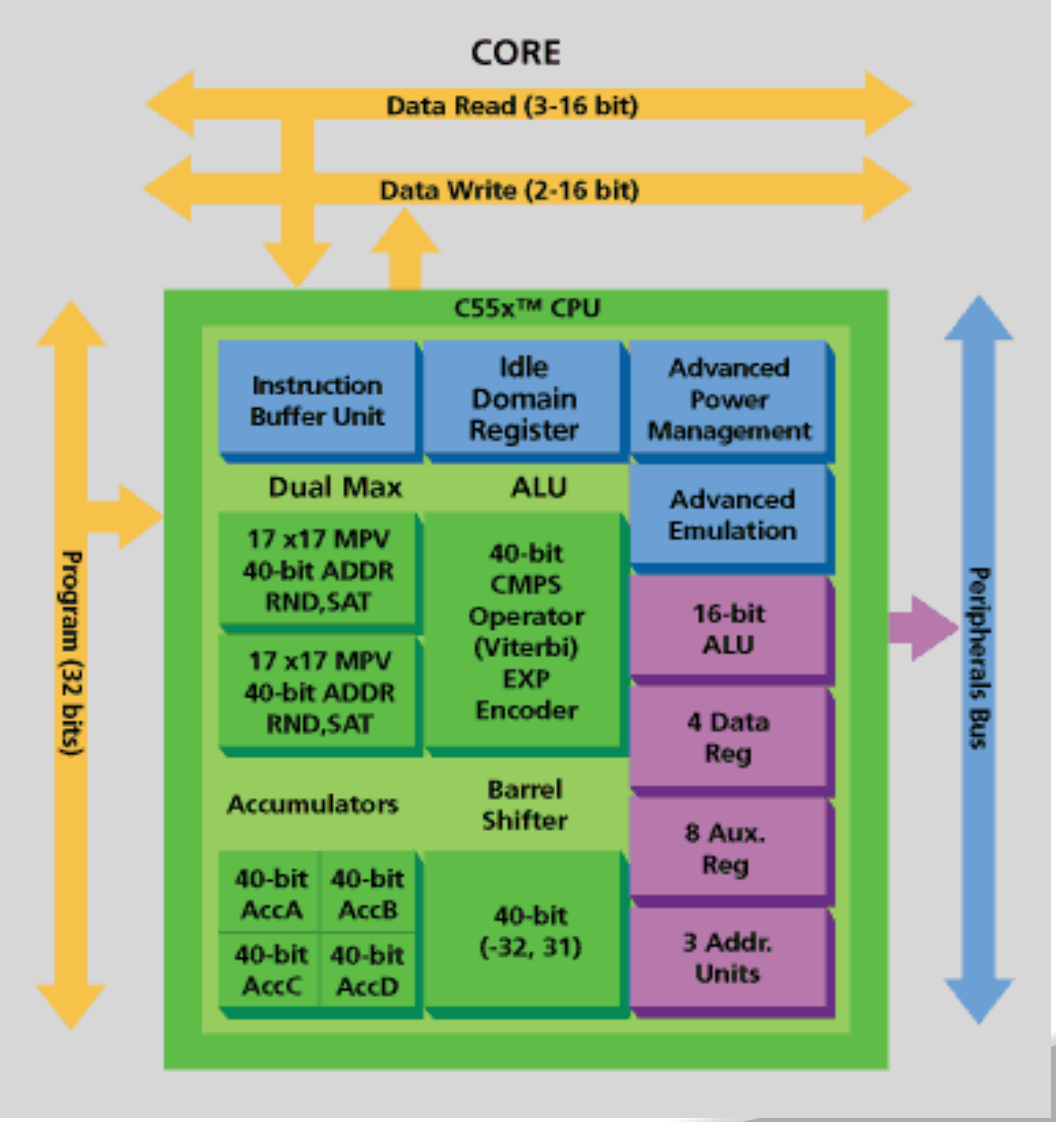
TI Processors, low power



TI, C64

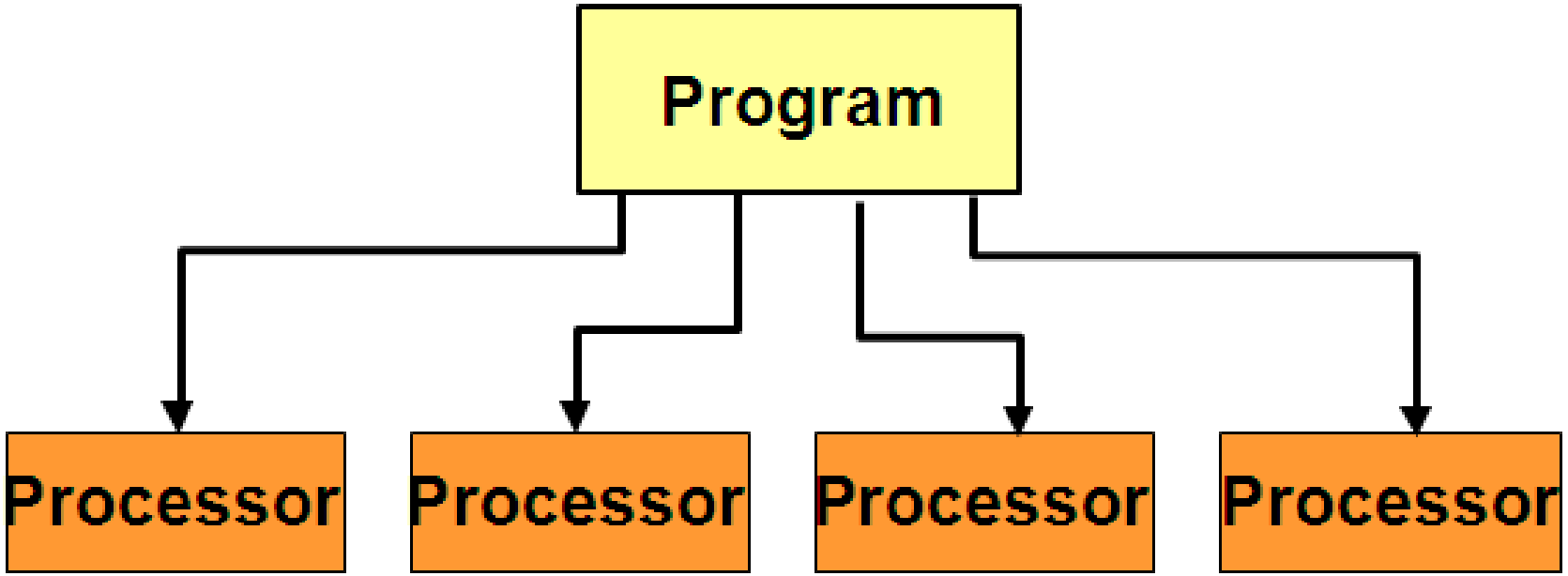

L1 Program Cache
C64x™ DSP CPU
Instruction Fetch
Instruction Dispatch
Packet Boundary
Span
Instruction Decode
Control
Registers
Advanced
Emulation
Interrupt
Control
Data Path 1
Data Path 2
Register File A
A15-A0
A31-A16
Register File B
B15-B0
B31-B16
L1
S1
M1
D1
D2
M2
S2
L2
Peripherals
Enhaced DMA Controller
L2 Cache/Memory
L1 Data Cache

TI, C55


CORE
Data Read (3-16 bit)
Data Write (2-16 bit)
C55x™ CPU
Instruction Buffer Unit
Idle Domain Register
Advanced Power Management
Dual Max
ALU
Advanced Emulation
17 x17 MPV 40-bit ADDR RND,SAT
17 x17 MPV 40-bit ADDR RND,SAT
40-bit CMPS Operator (Viterbi) EXP Encoder
16-bit ALU
4 Data Reg
Accumulators
Barrel Shifter
8 Aux. Reg
40-bit AccA
40-bit AccB
40-bit AccC
40-bit AccD
40-bit (-32, 31)
3 Addr. Units
Program (32 bits)
Peripherals Bus

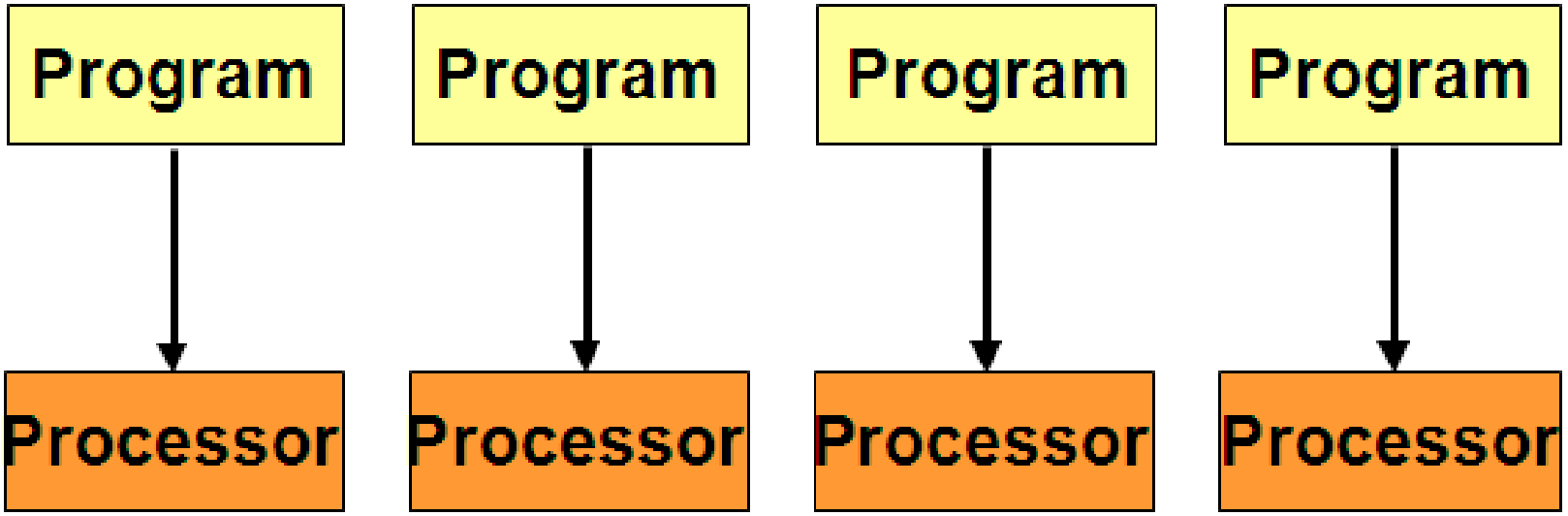
Processor Architectures

SIMD – Single Instruction Multiple Data



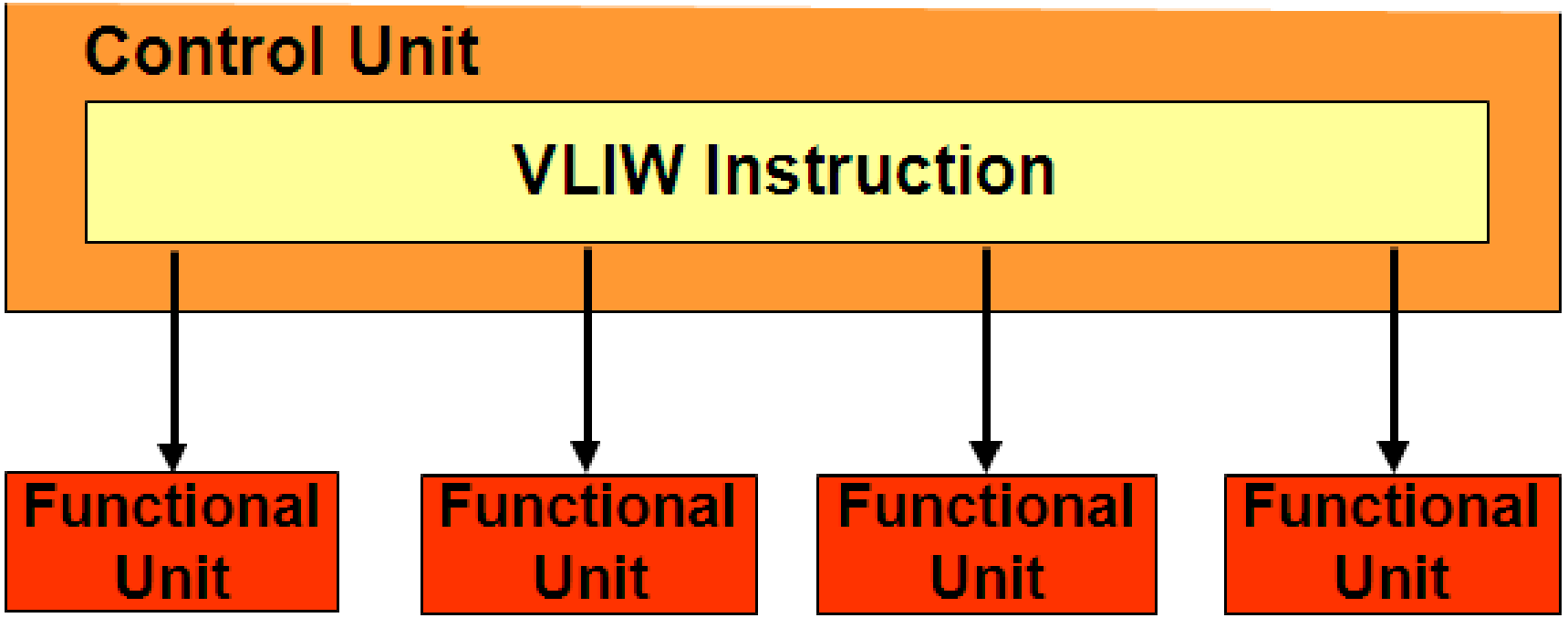
Processor Architectures Cont..

MIMD – Multiple Instruction Multiple Data

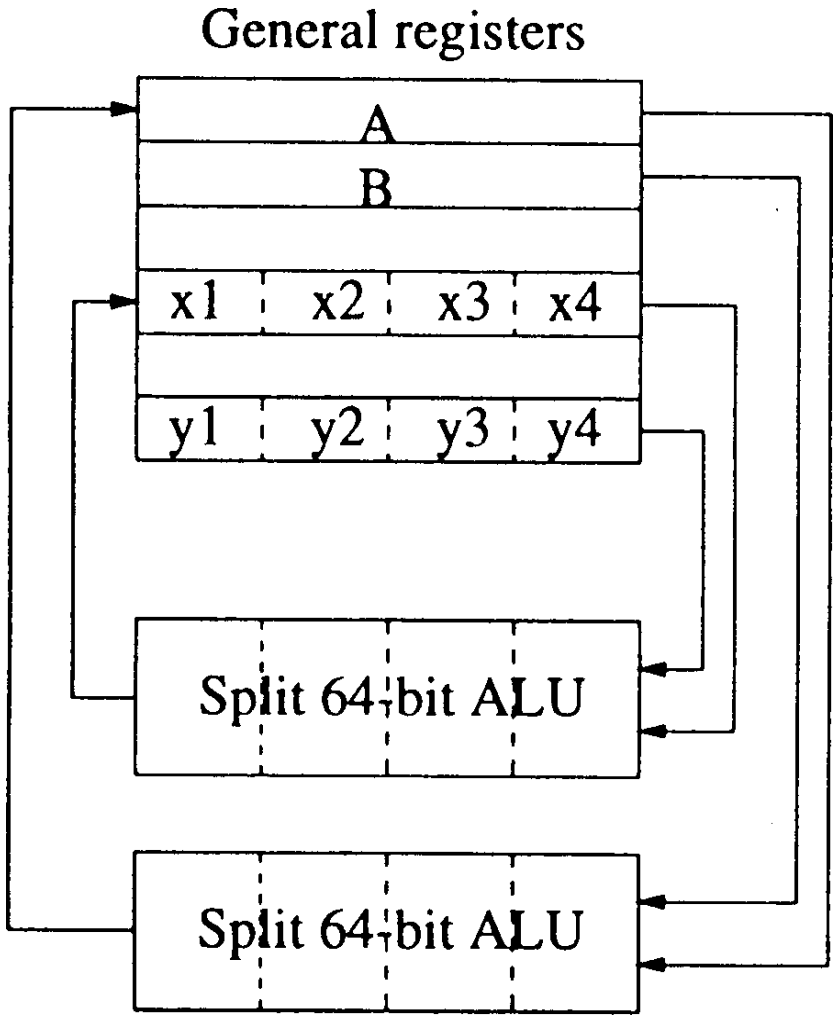


Processor Architectures Cont..

VLIW – Very Long Instruction Words

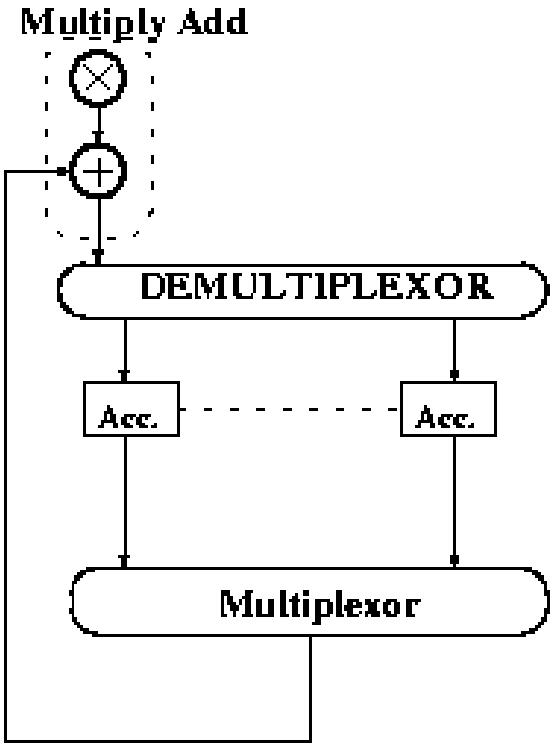


Split Processors



Functional units can be split into submodules, e.g. for images (8bits) TI320C80,
1 RISC
4 x 32bit DSP which can be split into 8bit modules

Low Power MMAC Multiplier Multiple Accumulator



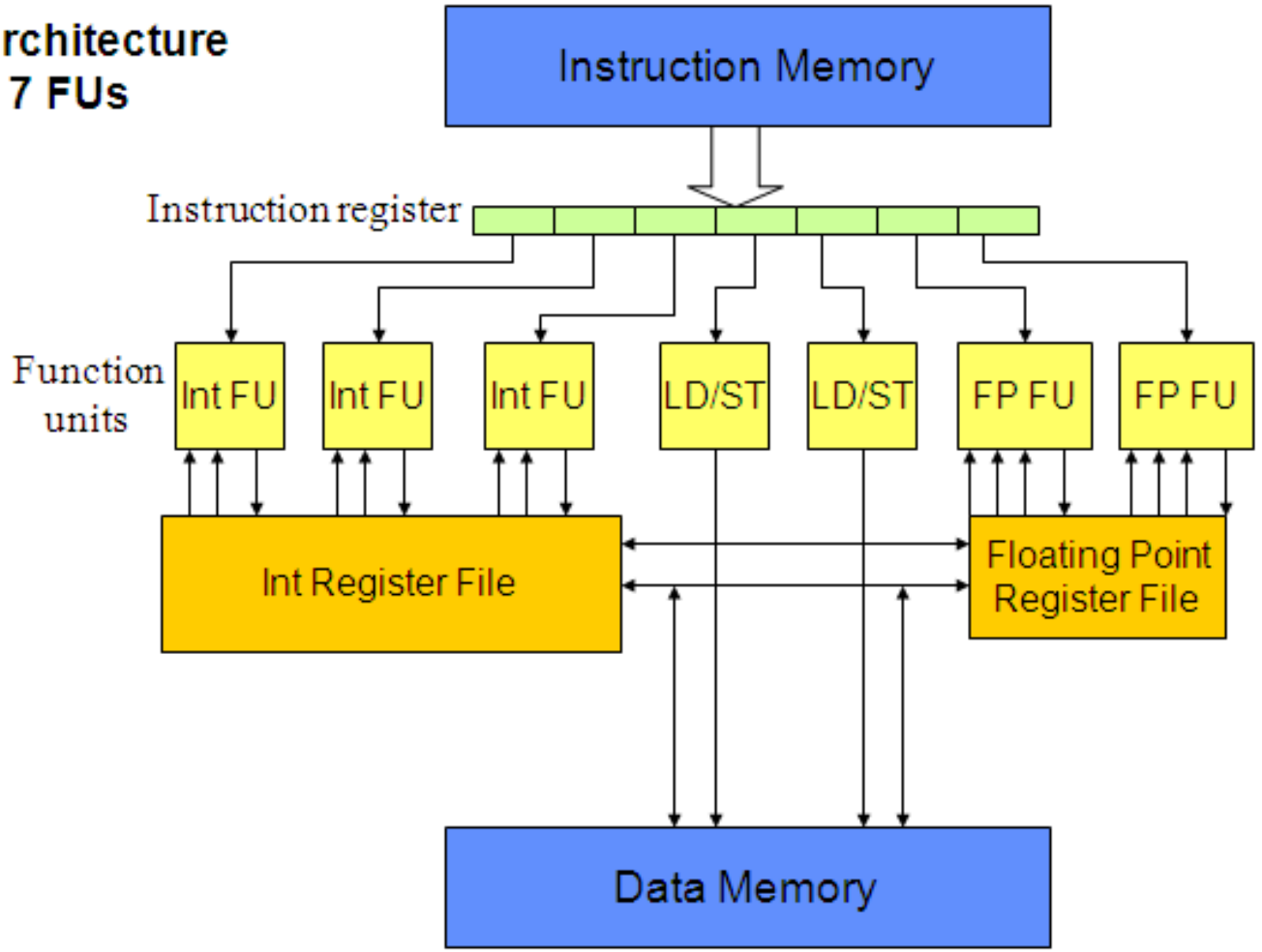
MMAC architecture: the number of output iterations that can coexist is equal to the number of Accumulators

- By using anti-aliasing filters.

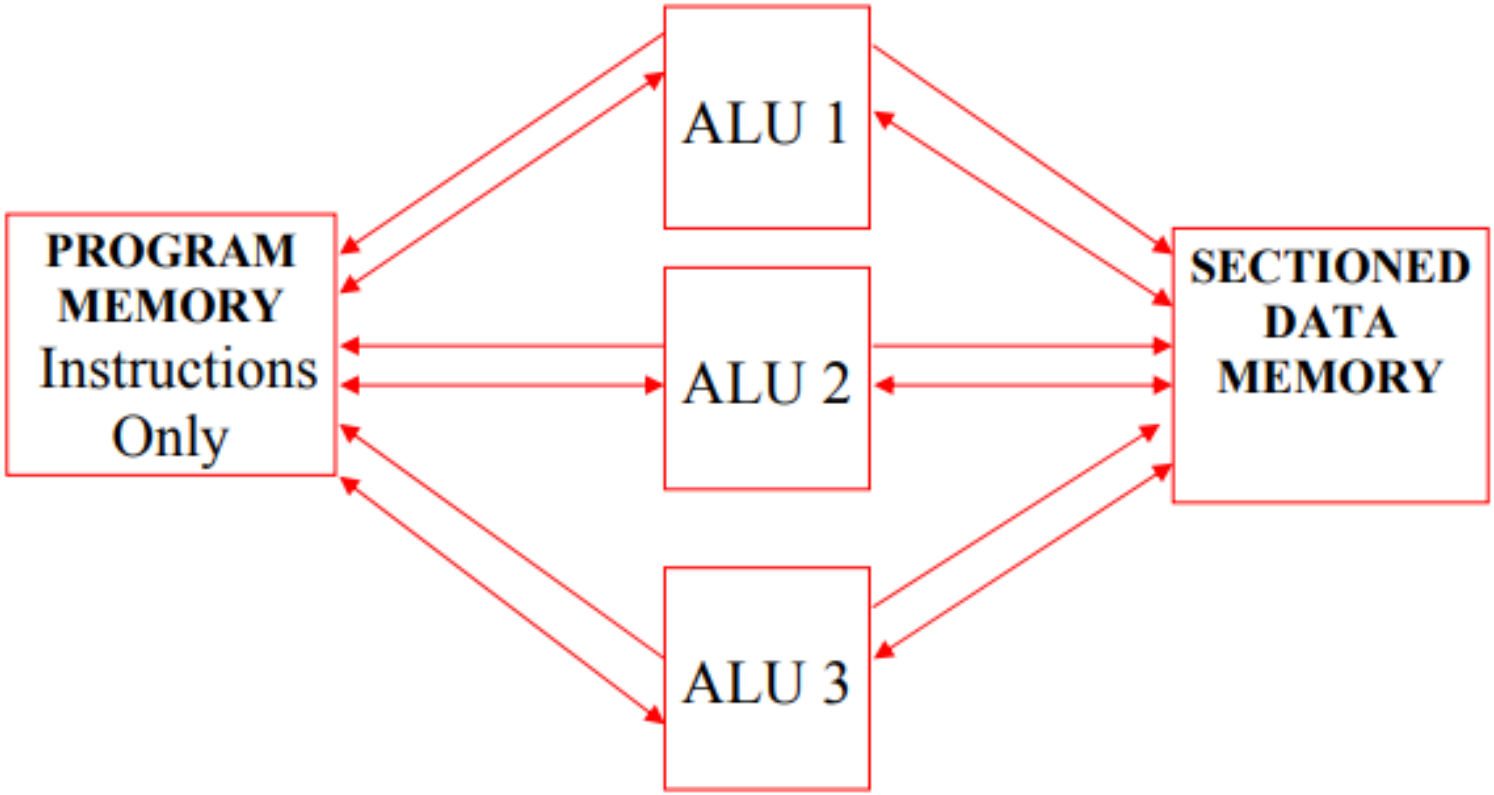
VLIW ARCHITECTURE

VLIW: general concept

A VLIW architecture with 7 FUs



Basic structure of VLIW Architecture



VLIW characteristics

- Multiple operations per instruction
- One instruction per cycle issued (at most)
- Compiler is in control
- Only RISC like operation support
  - Short cycle times
  - Easier to compile for
- Flexible: Can implement any FU mixture
- Extensible / Scalable

However:

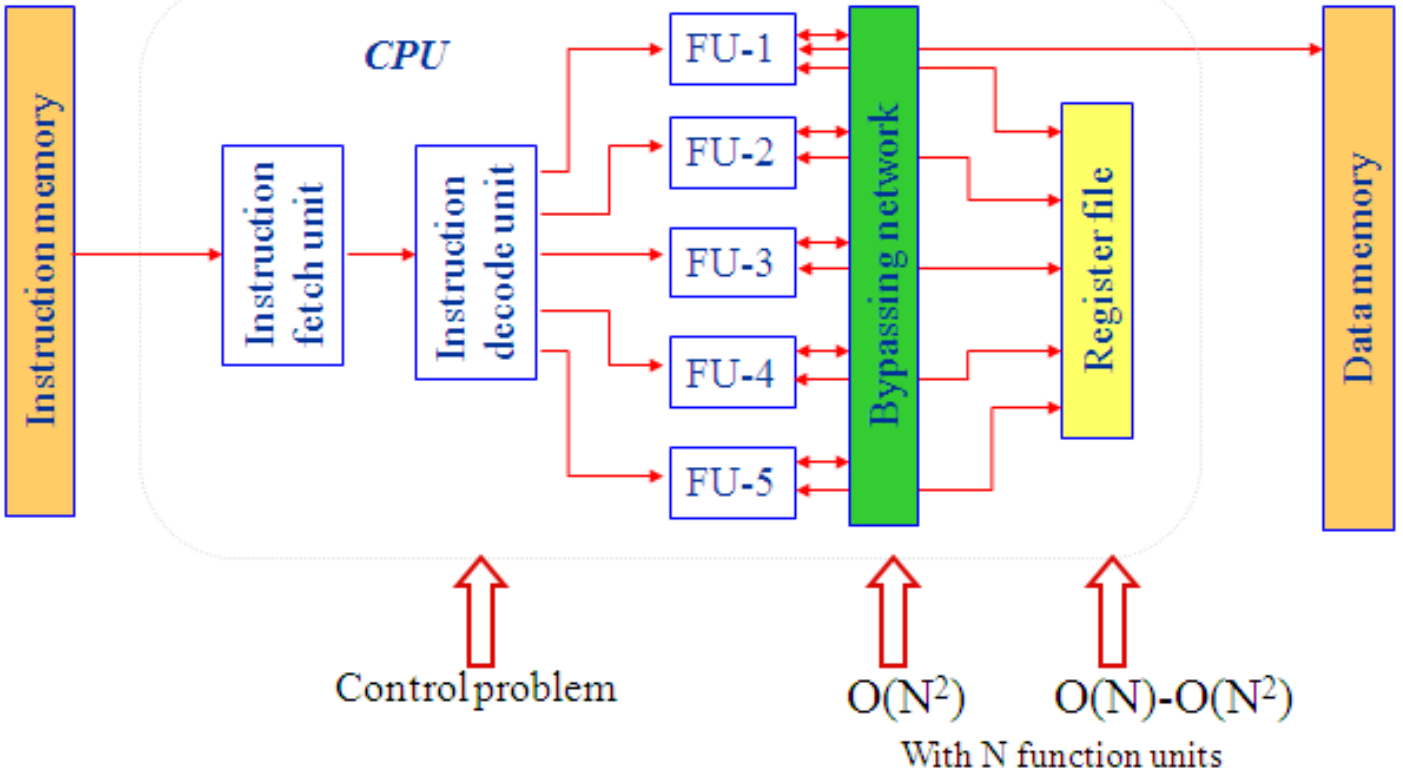
- tight inter FU connectivity required
- ***not binary compatible !!***
  - (new long instruction format)
- low code density

VLIW example: TMS320C62

TMS320C62 VelociTI Processor

- 8 operations (of 32-bit) per instruction (256 bit)
- Two clusters
  - 8 Fus; 4 Fus / cluster : (2 Multipliers, 6 ALUs)
  - 2 x 16 registers
  - One bus available to write in register file of other cluster
- Flexible addressing modes (like circular addressing)
- Flexible instruction packing
- All instruction conditional
- Originally: 5 ns, 200 MHz, 0.25 um, 5-layer CMOS
- 128 KB on-chip RAM

VLIW evaluation



VLIW evaluation

Strong points of VLIW:

- Scalable (add more FUs)
- Flexible (an FU can be almost anything; e.g. multimedia support)

Weak points:

- With N FUs:
  - Bypassing complexity: $O(N^2)$
  - Register file complexity: $O(N)$
  - Register file size: $O(N^2)$
- Register file design restricts FU flexibility

TMS320C6000 DSP Family Overview

- With a performance of up to 6000 million instructions per second (MIPS) and an efficient C compiler, the TMS320C6000 DSPs give system architects unlimited possibilities to differentiate their products.
- High performance, ease of use, and affordable pricing make the C6000 generation the ideal solution for multichannel, multifunction applications, such as:
- Pooled modems
- Wireless local loop base stations
- Remote access servers (RAS)
- Digital subscriber loop (DSL) systems
- Cable modems
- Multichannel telephony systems

Cont..

- All three DSP generations use the VelociTI architecture, a high-performance, advanced very long instruction word (VLIW) architecture, making these DSPs excellent choices for multichannel and multifunction applications.
- The TMS320C67x+ DSP is an enhancement of the C67x DSP with added functionality and an expanded instruction set.
- Any reference to the C67x DSP or C67x CPU also applies, unless otherwise noted, to the C67x+ DSP and C67x+ CPU, respectively

Standard DSP Alternatives

The C6000 generation is also an ideal solution for exciting new applications; for example:

- Personalized home security with face and hand/fingerprint recognition Advanced cruise control with global positioning systems (GPS) navigation and accident avoidance

- Remote medical diagnostics
- Beam-forming base stations
- Virtual reality 3-D graphics
- Speech recognition

Cont..

- Audio
- Radar
- Atmospheric modelling
- Finite element analysis
- Imaging (examples: fingerprint recognition, ultrasound, and MRI)

TMS320C6000 DSP Features and Options

Features of the C6000 devices include:

- Advanced VLIW CPU with eight functional units, including two multipliers and six arithmetic units
- Executes up to eight instructions per cycle for up to ten times the performance of typical DSPs
- Allows designers to develop highly effective RISC-like code for fast development time Instruction packing
- Gives code size equivalence for eight instructions executed serially or in parallel Reduces code size, program fetches, and power consumption

Cont..

Conditional execution of all instructions

- Reduces costly branching
- Increases parallelism for higher sustained performance
- Efficient code execution on independent functional units
- Industry's most efficient C compiler on DSP benchmark suite
- Industry's first assembly optimizer for fast development and improved parallelization
- 8/16/32-bit data support, providing efficient memory support for a variety of applications

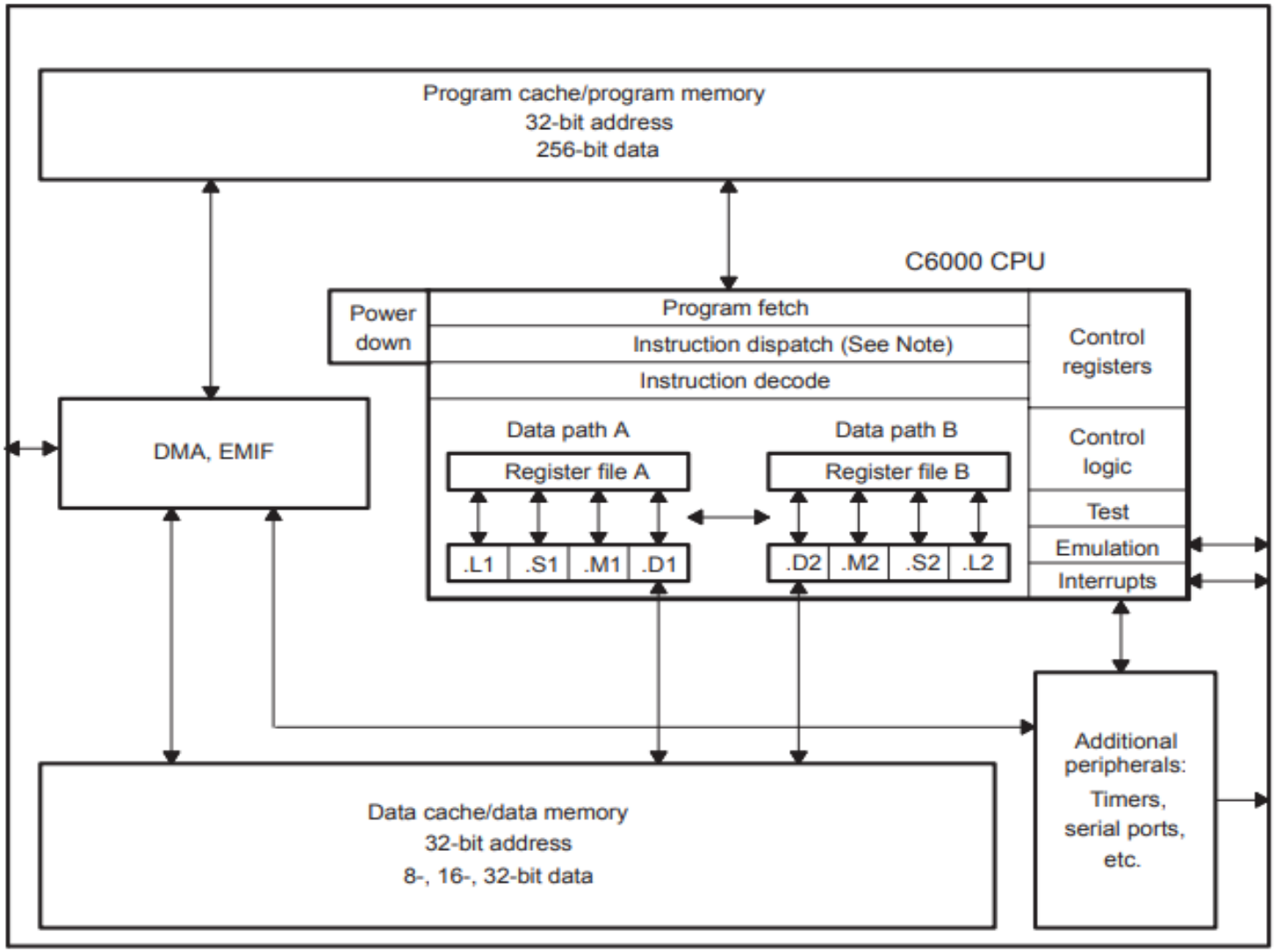
TMS320C6x Series

- The TMS320C6000 digital signal processor platform is part of the TMS320 DSP family.
- Fixed Point Devices
  - TMS320C62x DSP generation
  - TMS320C64x DSP generation
- Floating point devices
  - TMS320C67x DSP generation.
  - All three use the VelociTI architecture, a high-performance, advanced VLIW (very long instruction word) architecture
  - Excellent choices for multichannel and multifunction applications.

Cont..

- VelociTI's advanced features include
  - Instruction packing: reduced code size
    - Gives code size equivalence for eight instructions
    - Reduces code size, program fetches, and power consumption
  - Conditional execution of all instructions
    - Reduces costly branching
    - Increases parallelism for higher sustained performance
  - Fully pipelined branches: zero-overhead branching.

TMS320C67x DSP Architecture



Central Processing Unit (CPU)

The C67x CPU, in Figure 1−1, is common to all the C62x/C64x/C67x devices.

The CPU contains:

- ✓Program fetch unit
- ✓Instruction dispatch unit
- ✓Instruction decode unit
- ✓Two data paths, each with four functional units
- ✓32 32-bit registers
- ✓Control registers
- ✓Control logic
- ✓Test, emulation, and interrupt logic

Internal Memory

- The C67x DSP has a 32-bit, byte-addressable address space. Internal (on-chip) memory is organized in separate data and program spaces.
- When off-chip memory is used, these spaces are unified on most devices to a single memory space via the external memory interface (EMIF).
- The C67x DSP has two 32-bit internal ports to access internal data memory.
- The C67x DSP has a single internal port to access internal program memory, with an instruction-fetch width of 256 bits

Memory and Peripheral Options

A variety of memory and peripheral options are available for the C6000 platform:

- Large on-chip RAM, up to 7M bits
- Program cache
- 2-level caches
- 32-bit external memory interface supports SDRAM, SBSRAM, SRAM, and other asynchronous memories for a broad range of external memory requirements and maximum system performance.

General-Purpose Register Files

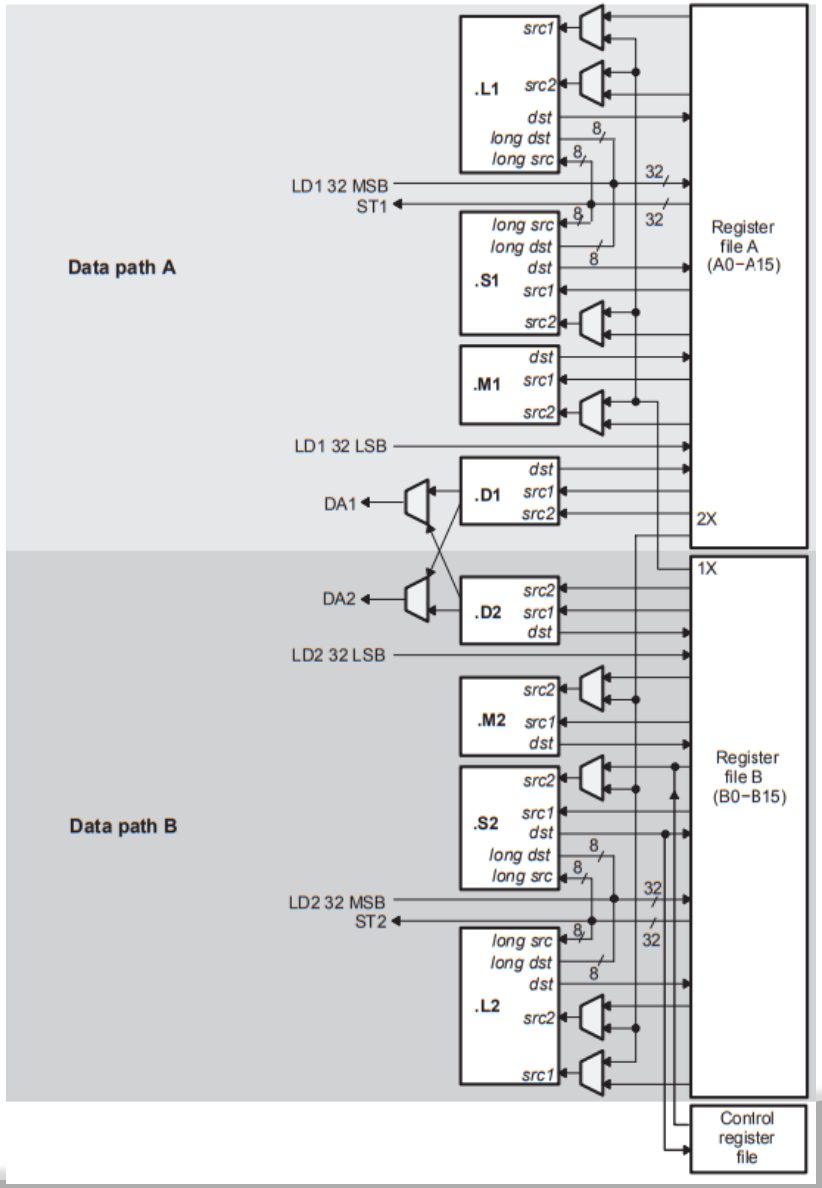
- There are two general-purpose register files (A and B) in the C6000 data paths. For the C67x DSP, each of these files contains 16 32-bit registers (A0–A15 for file A and B0–B15 for file B), as shown in Table 2−1.
- For the C67x+ DSP, the register file size is doubled to 32 32-bit registers (A0–A31 for file A and B0–B21 for file B), as shown in Table 2−1.
- The general-purpose registers can be used for data, data address pointers, or condition registers.

Data paths

Table 2−1. 40-Bit/64-Bit Register Pairs

Register Files		Devices
A	B	
A1:A0	B1:B0	C67x DSP
A3:A2	B3:B2	
A5:A4	B5:B4	
A7:A6	B7:B6	
A9:A8	B9:B8	
A11:A10	B11:B10	
A13:A12	B13:B12	
A15:A14	B15:B14	
A17:A16	B17:B16	C67x+ DSP only
A19:A18	B19:B18	
A21:A20	B21:B20	
A23:A22	B23:B22	
A25:A24	B25:B24	
A27:A26	B27:B26	
A29:A28	B29:B28	
A31:A30	B31:B30	

Data paths cont..



RISC Vs FISC

CISC -- High Code Density

- Fewer instructions needed to specify the algorithm

RISC -- Simpler to Design & Faster to Silicon

- Higher Performance -- smaller die size
- Lower power consumption
- Easier to develop compilers to take advantage of all features

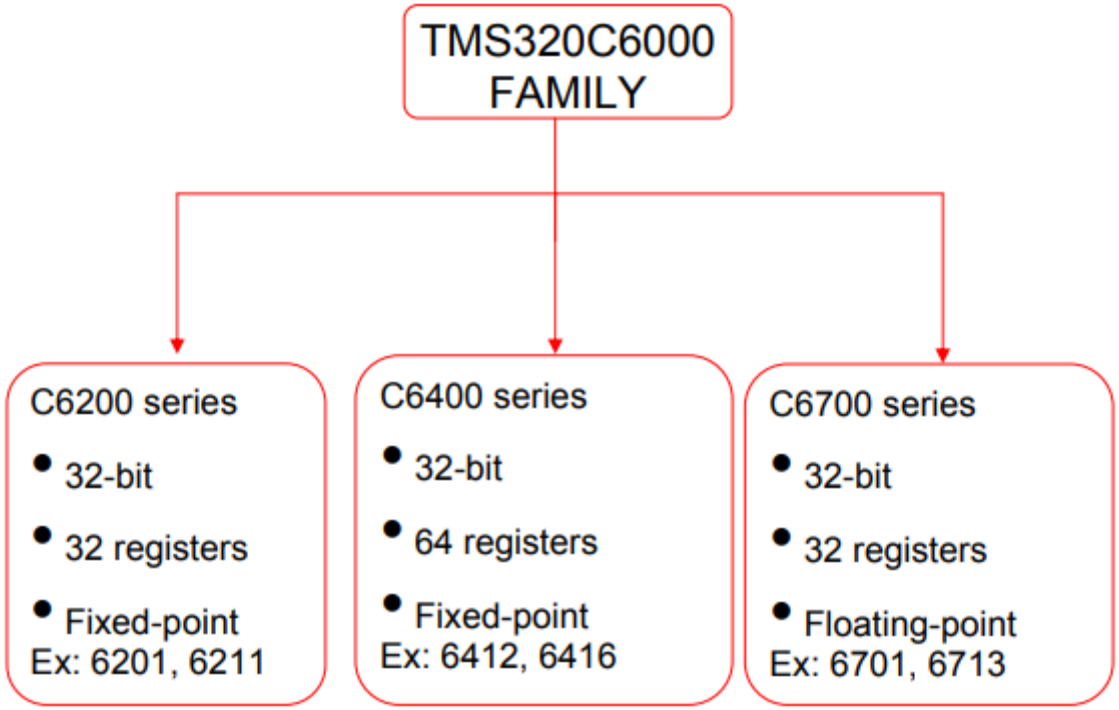
Register File Cross Paths

- Each functional unit reads directly from and writes directly to the register file within its own data path. That is, the .L1, .S1, .D1, and .M1 units write to register file A and the .L2, .S2, .D2, and .M2 units write to register file B.
- The register files are connected to the opposite-side register file's functional units via the 1X and 2X cross paths.
- These cross paths allow functional units from one data path to access a 32-bit operand from the opposite side register file.
- The 1X cross path allows the functional units of data path A to read their source from register file B, and the 2X cross path allows the functional units of data path B to read their source from register file A.

Register File Cross Paths

- On the C67x DSP, six of the eight functional units have access to the register file on the opposite side, via a cross path.
- The .M1, .M2, .S1, and .S2 units' src2 units are selectable between the cross path and the same side register file.
- In the case of the .L1 and .L2, both src1 and src2 inputs are also selectable between the cross path and the same-side register file.

TMS320C6000 FAMILY ARCHITECTURE



Instruction Set Mapping

Figure 13 shows the mapping between instruction set and the functional units.

.L Unit	.M Unit	.S Unit		.D unit
ABS	MPY	ADD	SET	ADD
ADD	MPYU	ADDK	SHL	ADDAB
ADDU	MPYUS	ADD2	SHR	ADDAH
AND	MPYSU	AND	SHRU	LDB
CMPEQ	MPYH	B disp	SSHL	LDBU
CMPGT	MPYHU	B IRP	SUB	LDH
CMPGTU	MPYHUS	B NRP	SUBU	LDHU
CMPLT	MPYHSU	B reg	SUB2	LDW
CMPLTU	MPYHL	CLR	XOR	MV
LMBD	MPYHLU	EXT	ZERO	STB
MV	MPYHULS	EXTU		STH
NEG	MPYHSLU	MV		STW
NORM	MYPLH	MVC		SUB
NOT	MPYLHU	MVK		SUBAB
OR	MPYLUHS	MVKH		SUBAH
SADD	MPYLSHU	MVKLH		SUBAW
SAT	SMPY	NEG		ZERO
SSUB	SMPYHL	NOT		
SUB	SMPYLH	OR		
SUBU	SMPYH			
SUBC				
XOR				
ZERO				

Figure 13: Instruction Set Mapping

Functional Units

The eight functional units in the C6000 data paths can be divided into two groups of four; each functional unit in one data path is almost identical to the corresponding unit in the other data path.

Functional Unit	Fixed-point Operations	Floating-point Operations
.L unit (.L1 and .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits **Byte shifts** **Data packing/unpacking** **5-bit constant generation** **Dual 16-bit arithmetic** **Quad 8-bit arithmetic** **Dual 16-bit min/max** **Quad 8-bit min/max**	Arithmetic operations DP→SP, INT→DP, INT→SP conversion operations
.S unit (.S1 and .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) **Byte shifts** **Data packing/unpacking** **Dual 16-bit compare** **Quad 8-bit compare** **Dual 16-bit shift** **Dual 16-bit saturated arithmetic** **Quad 8-bit saturated arithmetic**	Compare Reciprocal and reciprocal square-root Operations Absolute value operations SP→DP conversion operations

FUNCTIONAL UNIT Cont...

.M unit (.M1 and .M2)	16 x 16 multiply operations **16 x 32 multiply operations** **Quad 8 x 8 multiply operations** **Dual 16 x 16 multiply operations** **Dual 16 x 16 multiply with add/subtract operations** **Quad 8 x 8 multiply with add** **Bit expansion** **Bit interleaving/de-interleaving** **Variable shift operations** **Rotation** **Galois Field Multiply**	32 x 32-bit fixed-point multiply operations Floating-point multiply operations
.D unit (.D1 and .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit offset Loads and stores with 15-bit offset **Load and store double words with 5-bit constant** **Load and store non-aligned words and double words** **5-bit constant generation** **32-bit logical operations**	Load doubleword with 5-bit constant offset

Memory Load Store Paths

- Two 32–bit paths for loading data from memory to the register file
  - LD1 (LD1 LSB and LD1 MSB) for register file A
  - LD2 (LD2 LSB and LD2 MSB) for register file B
- LDDW instruction simultaneously load two 32–bit values into register file A and two 32–bit values into register file B

Memory Store Paths

- Two 32–bit paths for storing data to memory from the register file
  - ST1 for register file A
  - ST2 for register file B

Data Address Paths

- DA1 & DA2 are connected to the .D units in both data paths.
- The DA1 and DA2 resources and their associated data paths are specified as T1 and T2 respectively.

Types of Addressing Modes

The addressing modes on the C62x, C64x, and C67x are

- Linear mode
- Circular mode using BK0
- Circular mode using BK1

Cont..

Register Addressing Mode: The operand is the contents of a processor register; the name of the register is given in the instruction.

Ex: ADD .L1 A1, A2, A3 ; A1 + A2 = A3
SUB .L2 B1, B2, B6 ; B1 – B2 = B6

Note that the functional unit is a must in writing assembly instructions.

Cont..

Immediate Addressing Mode: The operand is a numeric constant which is directly specified in the instruction.

Ex: ADD .L1 A1, 20, A3 ; A1 + 20 = A3

SUB .L2 B1, 15, B6 ; B1 – 15 = B6

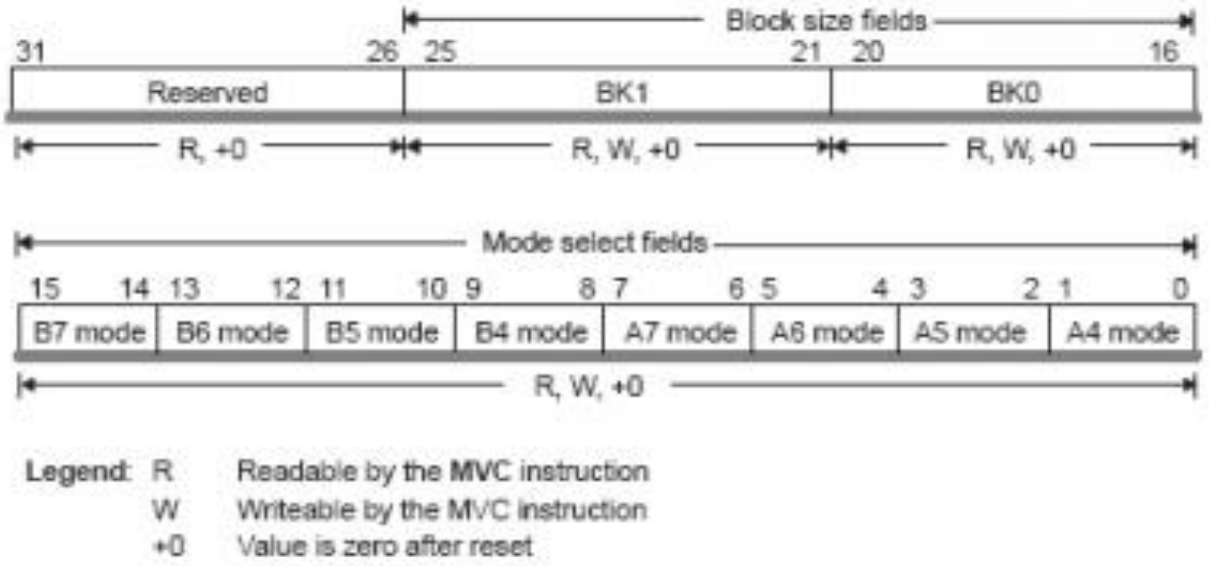
Indirect Addressing Mode: The effective address of the operand is the contents of a register that appears in the instruction. An asterisk (*) is used as an indirection operator. Also increment (++) and decrement (- -) operators are supported.

Ex: LDW .D2 *B0, B1 STW .D1 A1,*A2++

Addressing Mode Register

For each of the eight registers (A4–A7, B4–B7) that can perform linear or circular addressing, the AMR specifies the addressing mode. A 2-bit field for each register selects the address modification mode: linear (the default) or circular mode.

Addressing Mode Register (AMR)



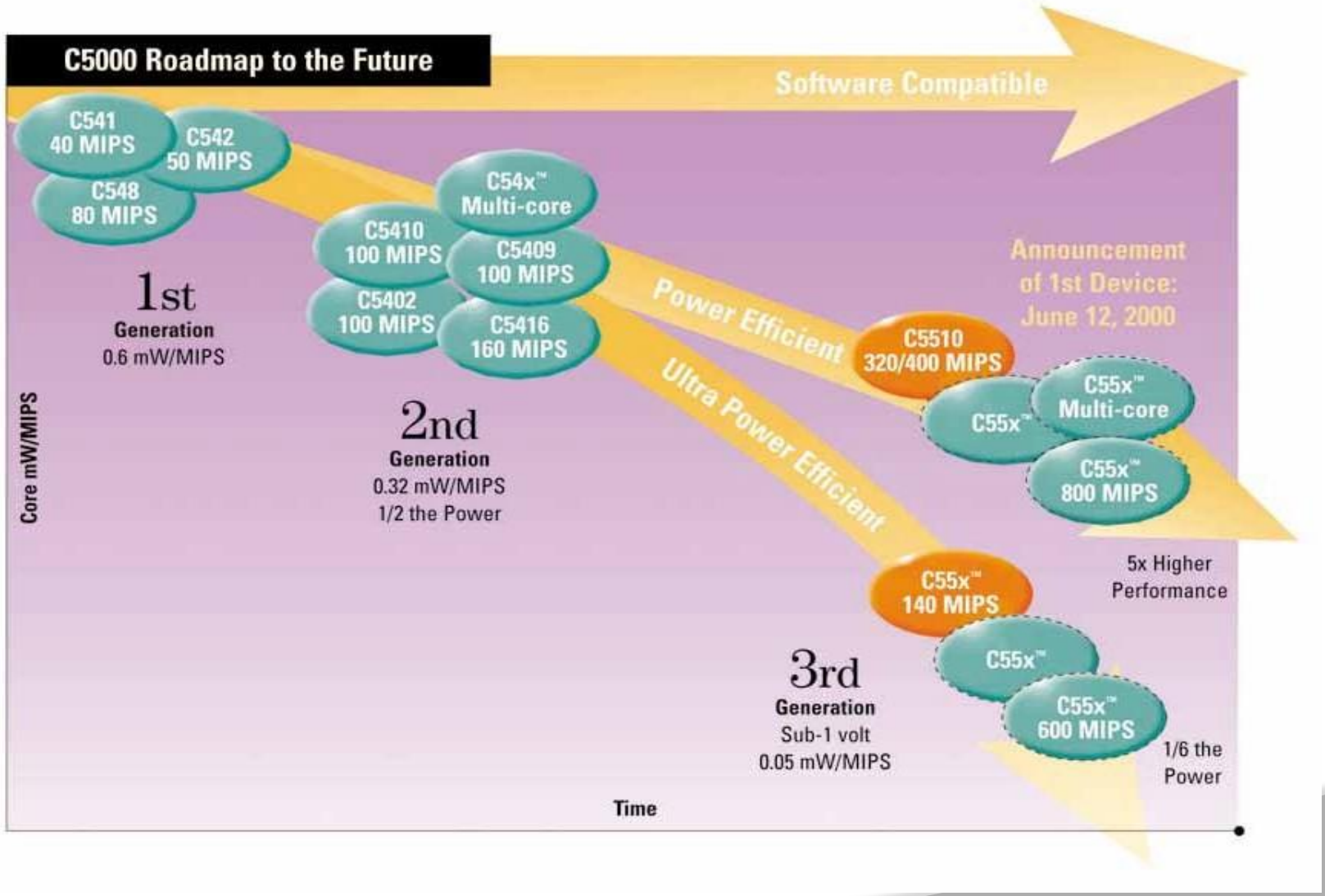
Mode Select Field Encoding

The reserved portion of AMR is always 0. The AMR is initialized to 0 at reset. The block size fields, BK0 and BK1, contain 5-bit values used in calculating block sizes for circular addressing.

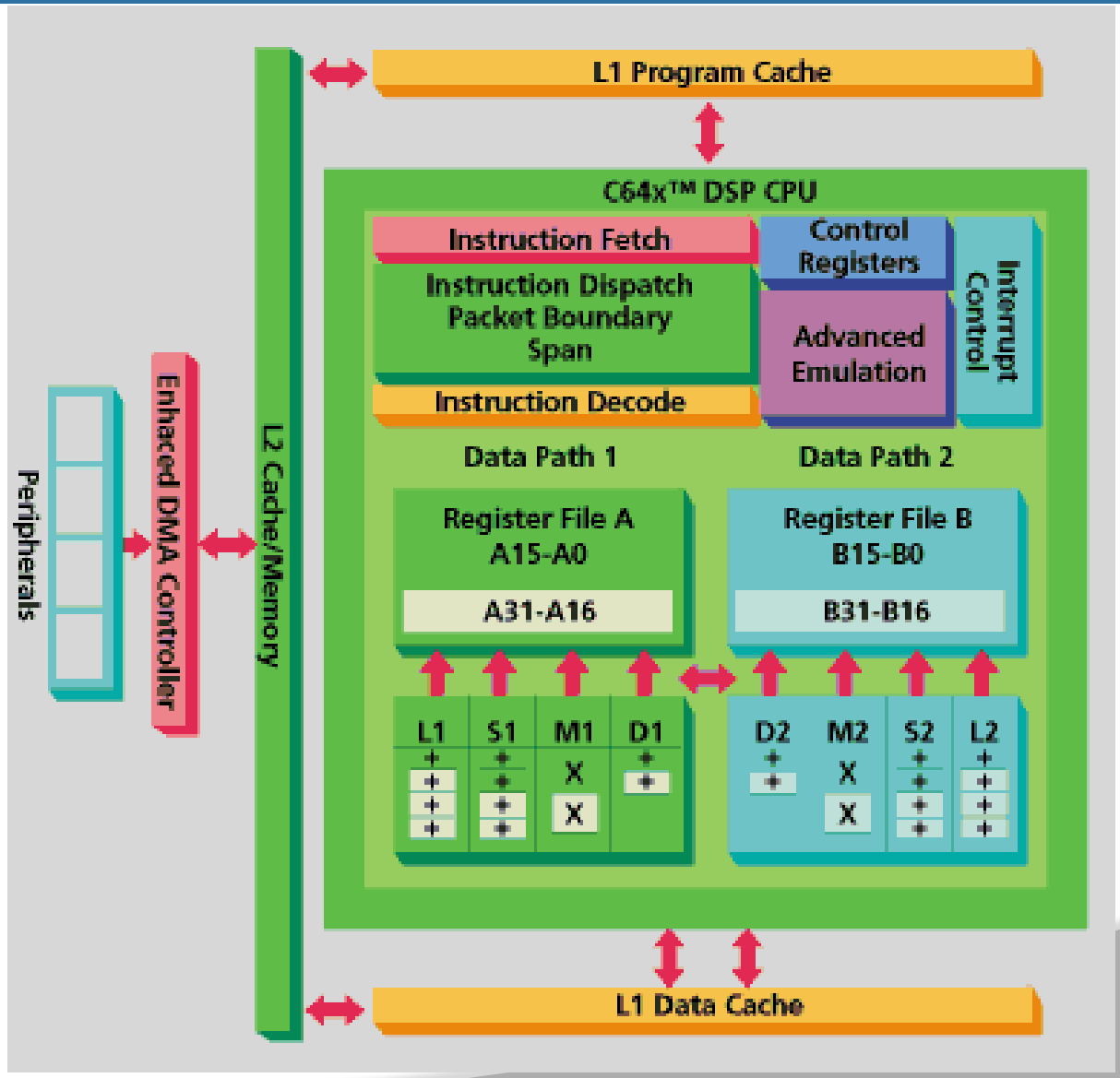
$$\text{Block size (in bytes)} = 2^{(N+1)}$$

Mode	Description
00	Linear mode (default at reset)
01	Circular Addressing using BK0 field
10	Circular Addressing using BK1 field
11	Reserved

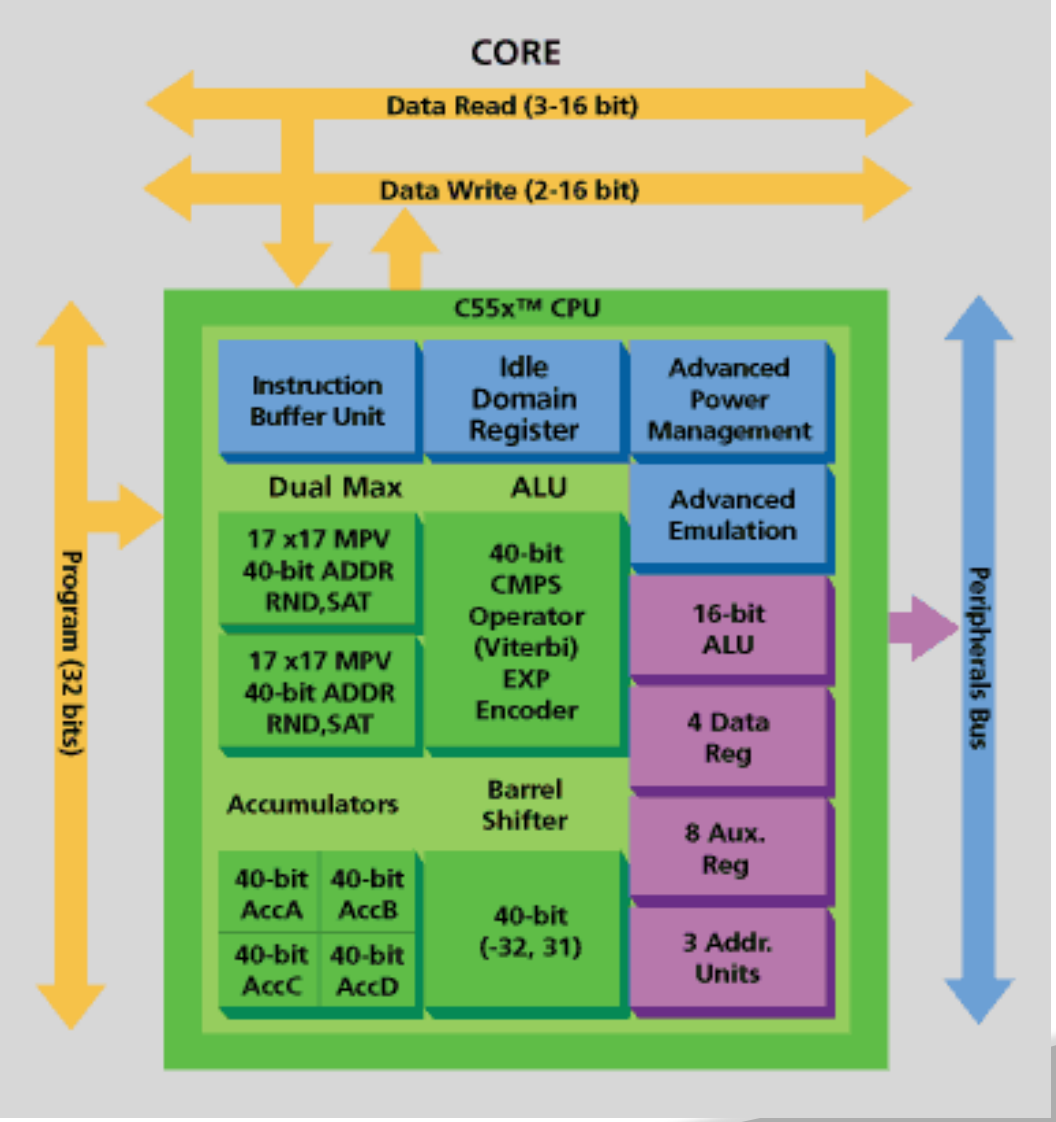
TI Processors, low power



TI, C64

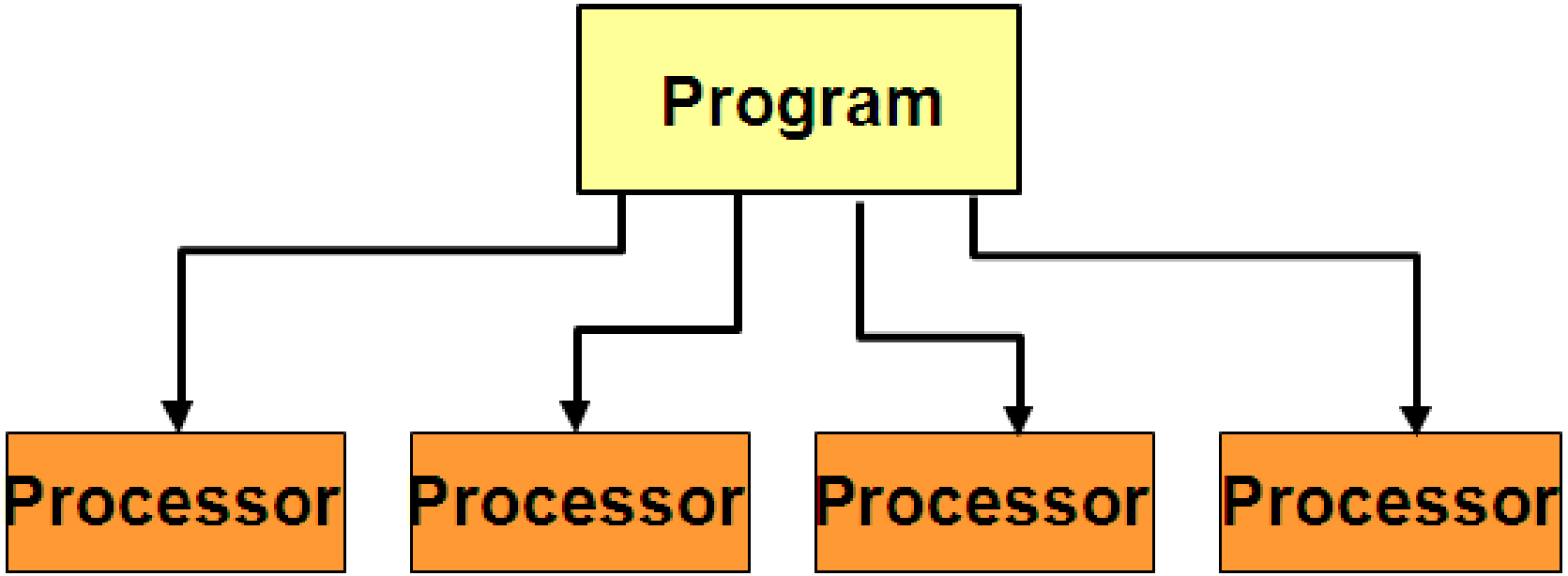

L1 Program Cache
C64x™ DSP CPU
Instruction Fetch
Instruction Dispatch
Packet Boundary
Span
Instruction Decode
Control
Registers
Advanced
Emulation
Interrupt
Control
Data Path 1
Data Path 2
Register File A
A15-A0
A31-A16
Register File B
B15-B0
B31-B16
L1
S1
M1
D1
D2
M2
S2
L2
Peripherals
Enhaced DMA Controller
L2 Cache/Memory
L1 Data Cache

TI, C55


CORE
Data Read (3-16 bit)
Data Write (2-16 bit)
C55x™ CPU
Instruction Buffer Unit
Idle Domain Register
Advanced Power Management
Dual Max
ALU
Advanced Emulation
17 x17 MPV 40-bit ADDR RND,SAT
17 x17 MPV 40-bit ADDR RND,SAT
40-bit CMPS Operator (Viterbi) EXP Encoder
16-bit ALU
4 Data Reg
Accumulators
Barrel Shifter
8 Aux. Reg
40-bit AccA
40-bit AccB
40-bit AccC
40-bit AccD
40-bit (-32, 31)
3 Addr. Units
Program (32 bits)
Peripherals Bus

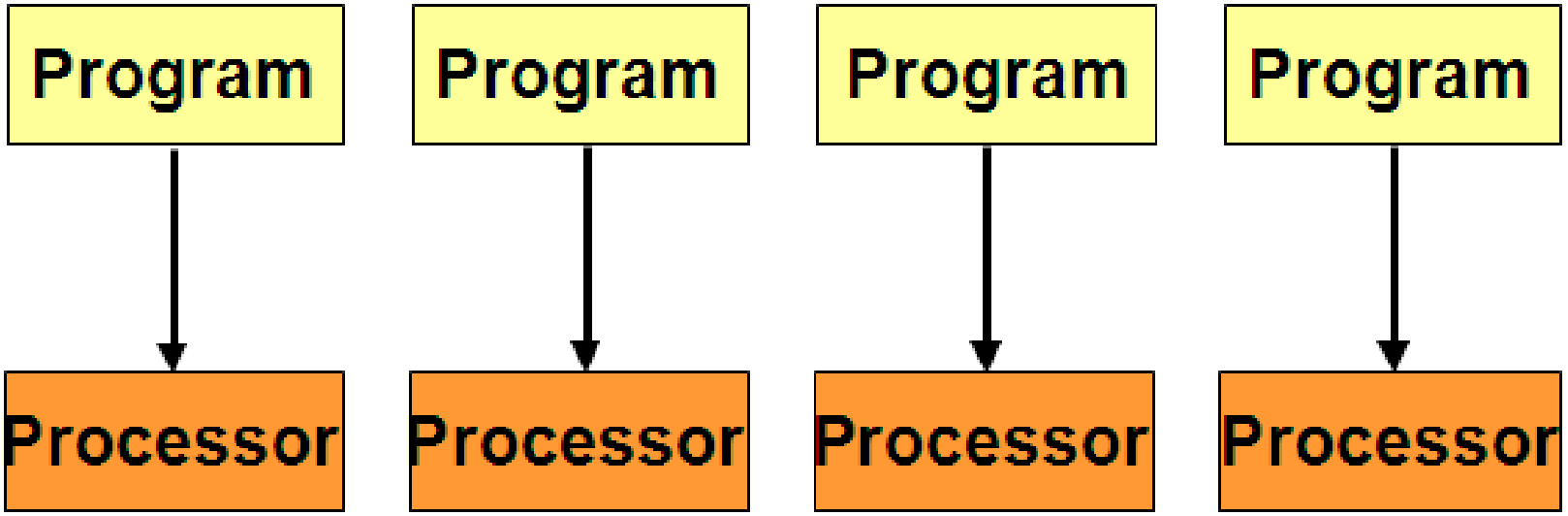
Processor Architectures

SIMD – Single Instruction Multiple Data



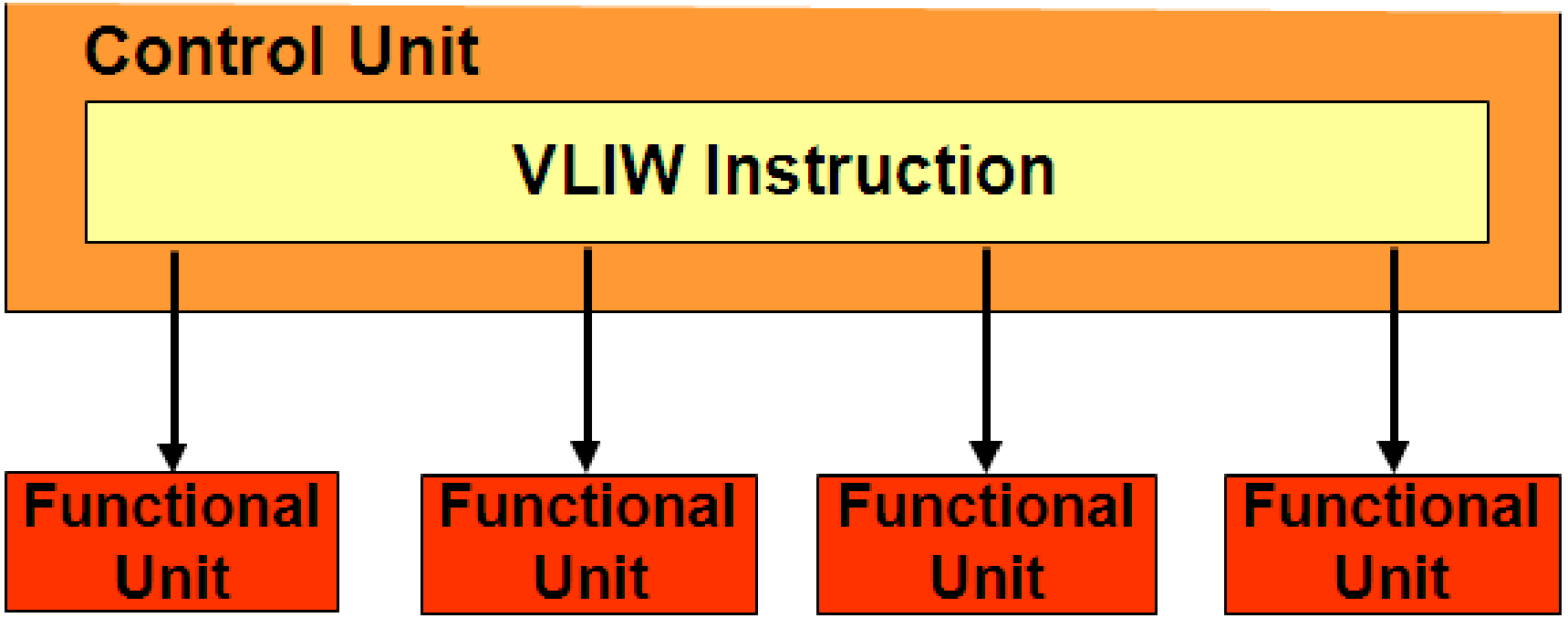
Processor Architectures Cont..

MIMD – Multiple Instruction Multiple Data

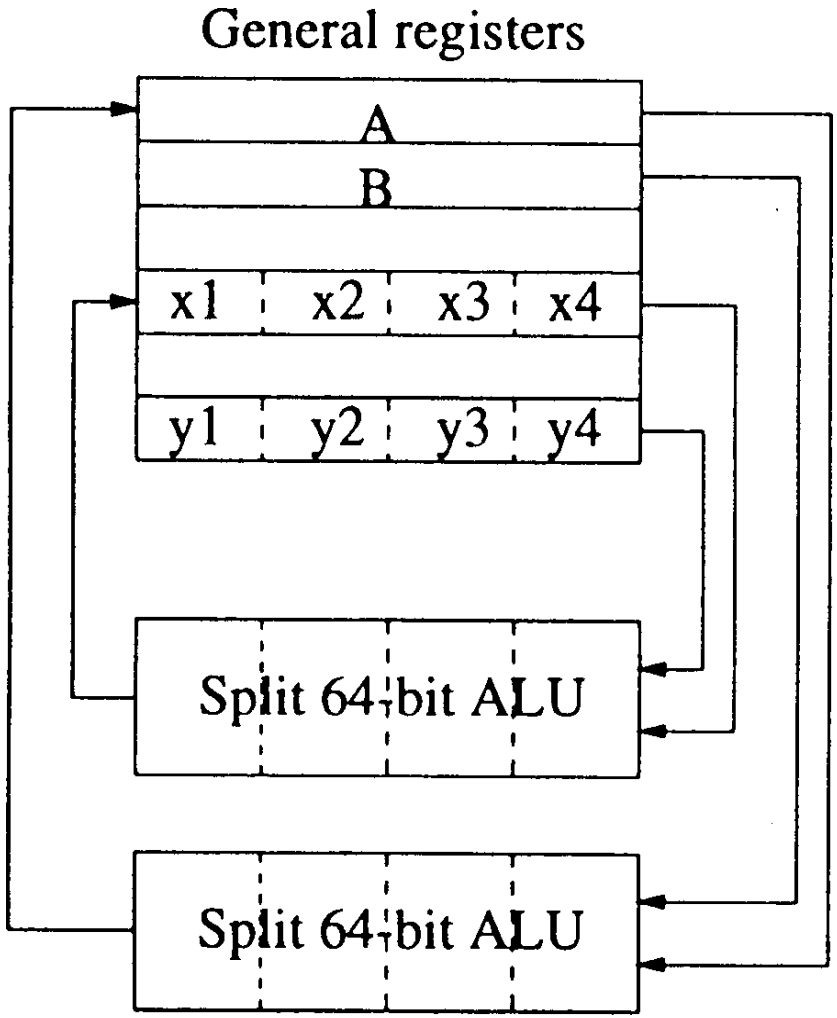


Processor Architectures Cont..

VLIW – Very Long Instruction Words

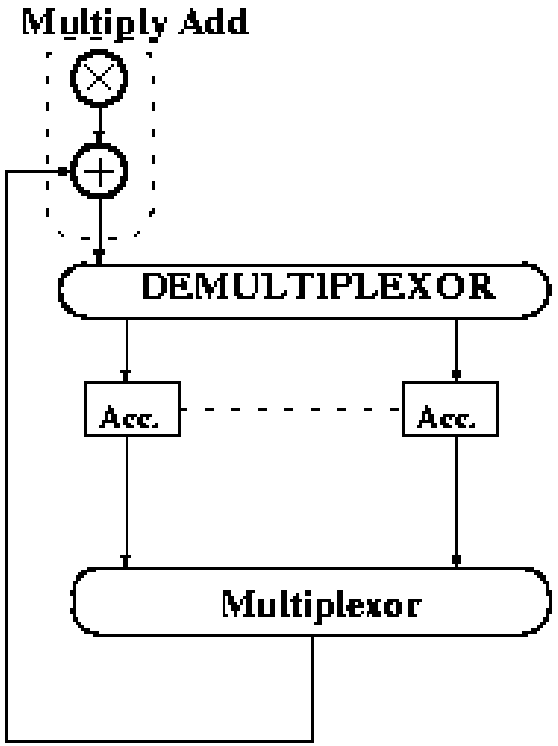


Split Processors



Functional units can be split into submodules, e.g. for images (8bits) TI320C80,
1 RISC
4 x 32bit DSP which can be split into 8bit modules

Low Power MMAC Multiplier Multiple Accumulator



MMAC architecture: the number of output iterations that can coexist is equal to the number of Accumulators

- By using anti-aliasing filters.