Lecture 9: Digital Signal Processors: Applications and Architectures

Prepared by: Professor Kurt Keutzer

Computer Science 252, Spring 2000

With contributions from:

Dr. Jeff Bier, BDTI; Dr. Brock Barton, TI;

Prof. Bob Brodersen, Prof. David Patterson

Processor Applications

General Purpose - high performance

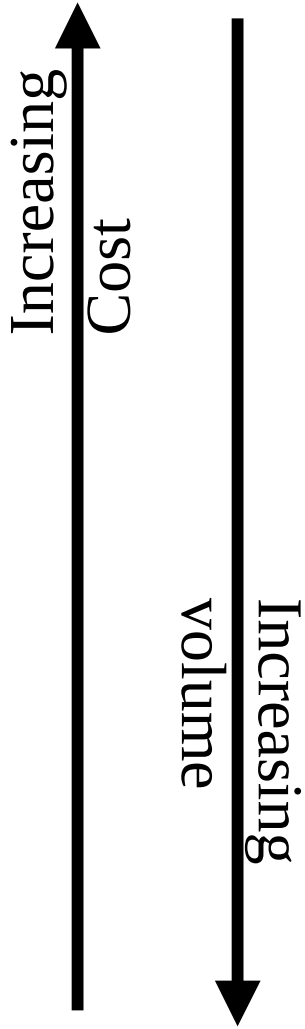
- **Pentiums, Alpha's, SPARC**
- **Used for general purpose software**
- **Heavy weight OS - UNIX, NT**
- **Workstations, PC's**

Embedded processors and processor cores

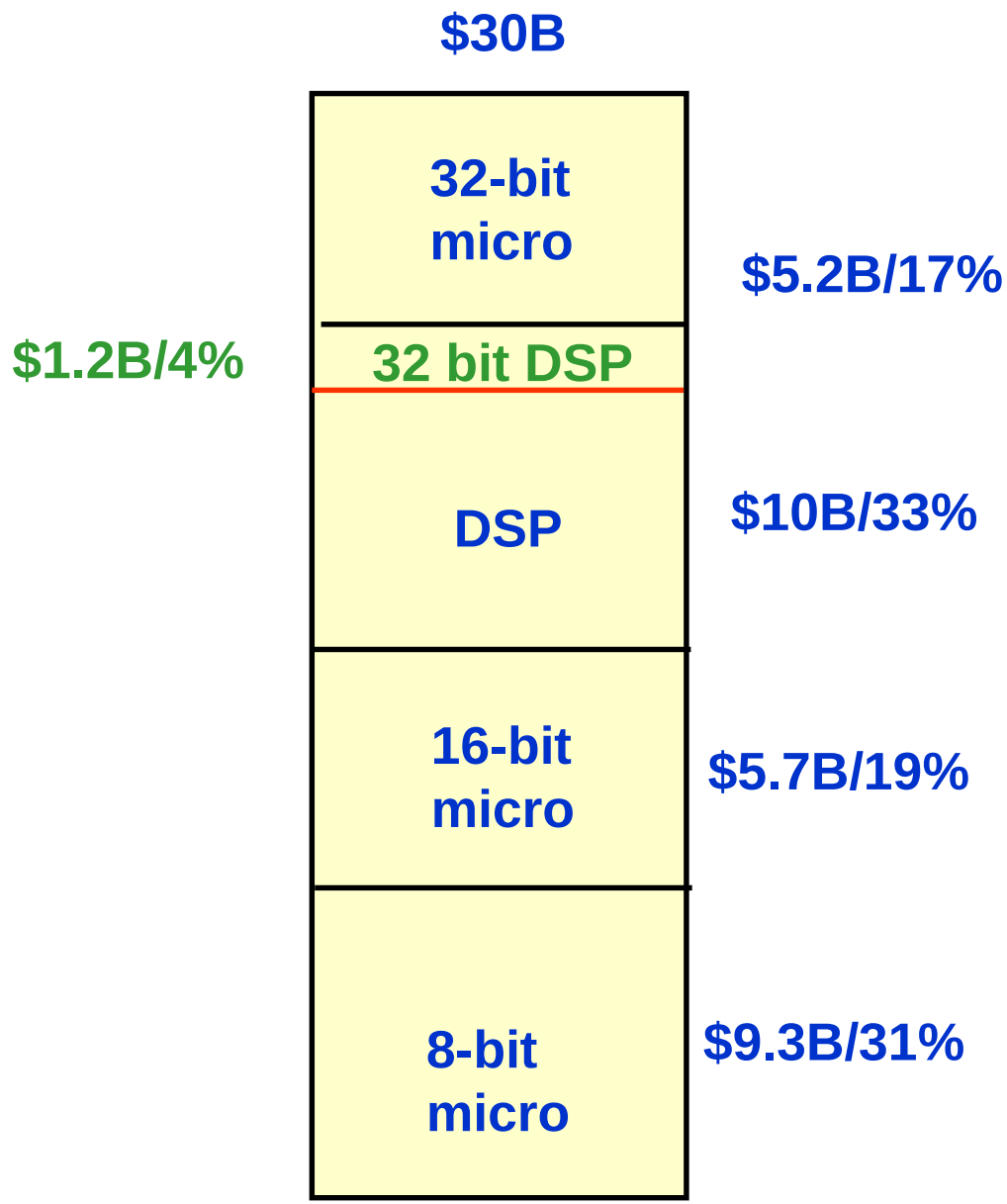
- **ARM, 486SX, Hitachi SH7000, NEC V800**
- **Single program**
- **Lightweight, often realtime OS**
- **DSP support**
- **Cellular phones, consumer electronics (e.g. CD players)**

Microcontrollers

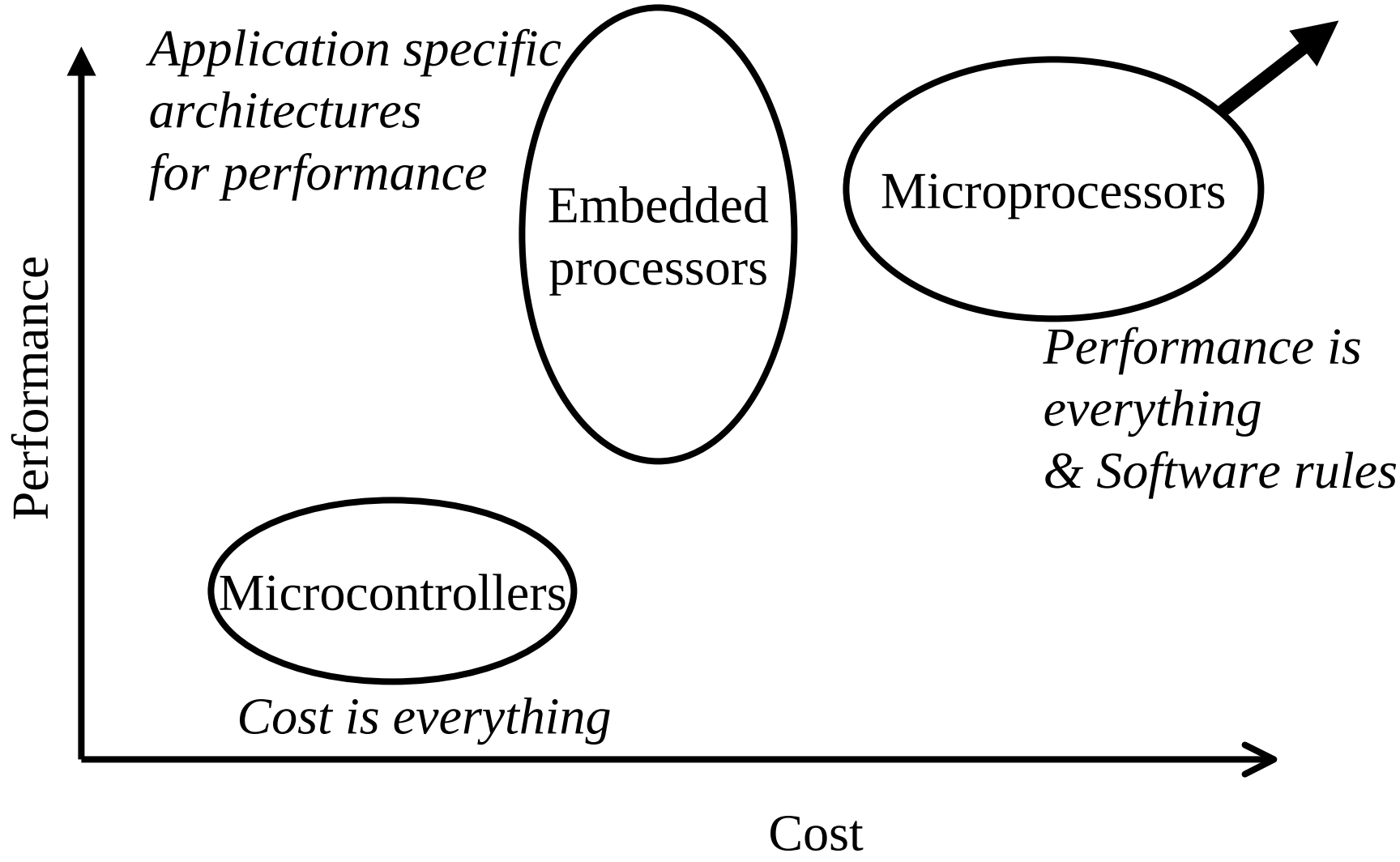
- **Extremely cost sensitive**
- **Small word size - 8 bit common**
- **Highest volume processors by far**
- **Automobiles, toasters, thermostats, ...**



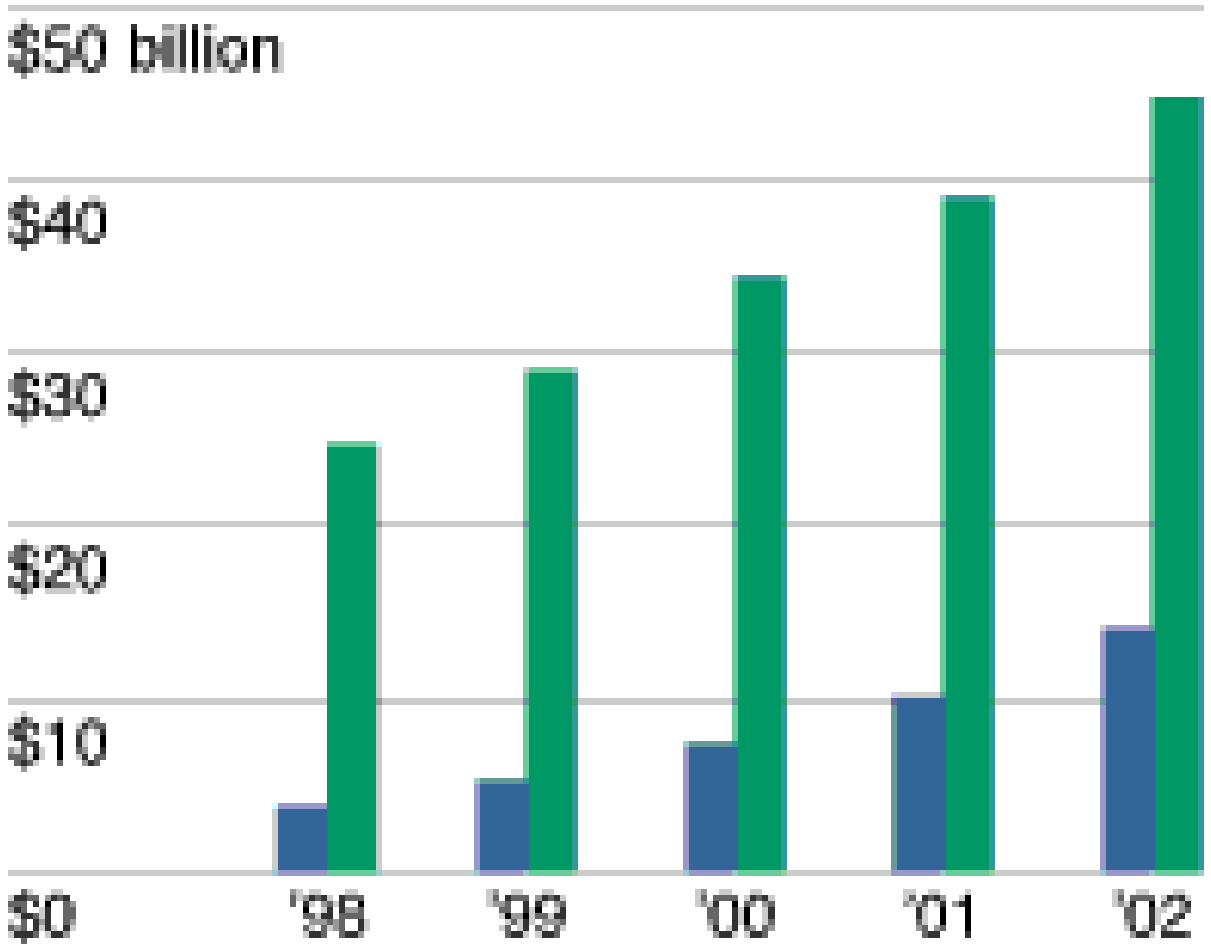
Processor Markets



The Processor Design Space



Market for DSP Products



DSP is the fastest growing segment of the semiconductor market

DSP Applications

Audio applications

- **MPEG Audio**
- **Portable audio**

Digital cameras

Wireless

- **Cellular telephones**
- **Base station**

Networking

- **Cable modems**
- **ADSL**
- **VDSL**

Another Look at DSP Applications

High-end

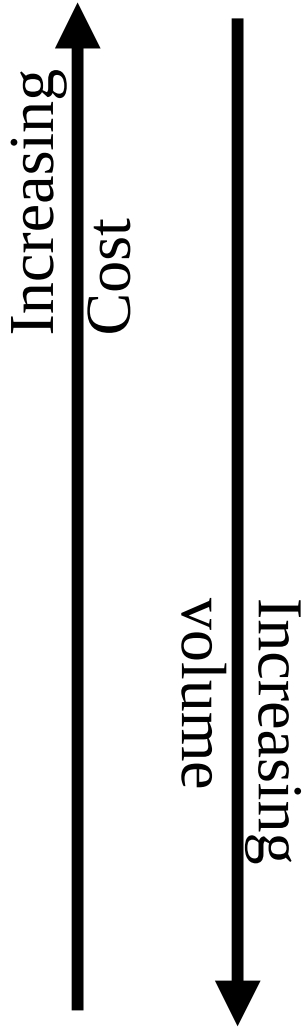
- **Wireless Base Station - TMS320C6000**
- **Cable modem**
- **gateways**

Mid-end

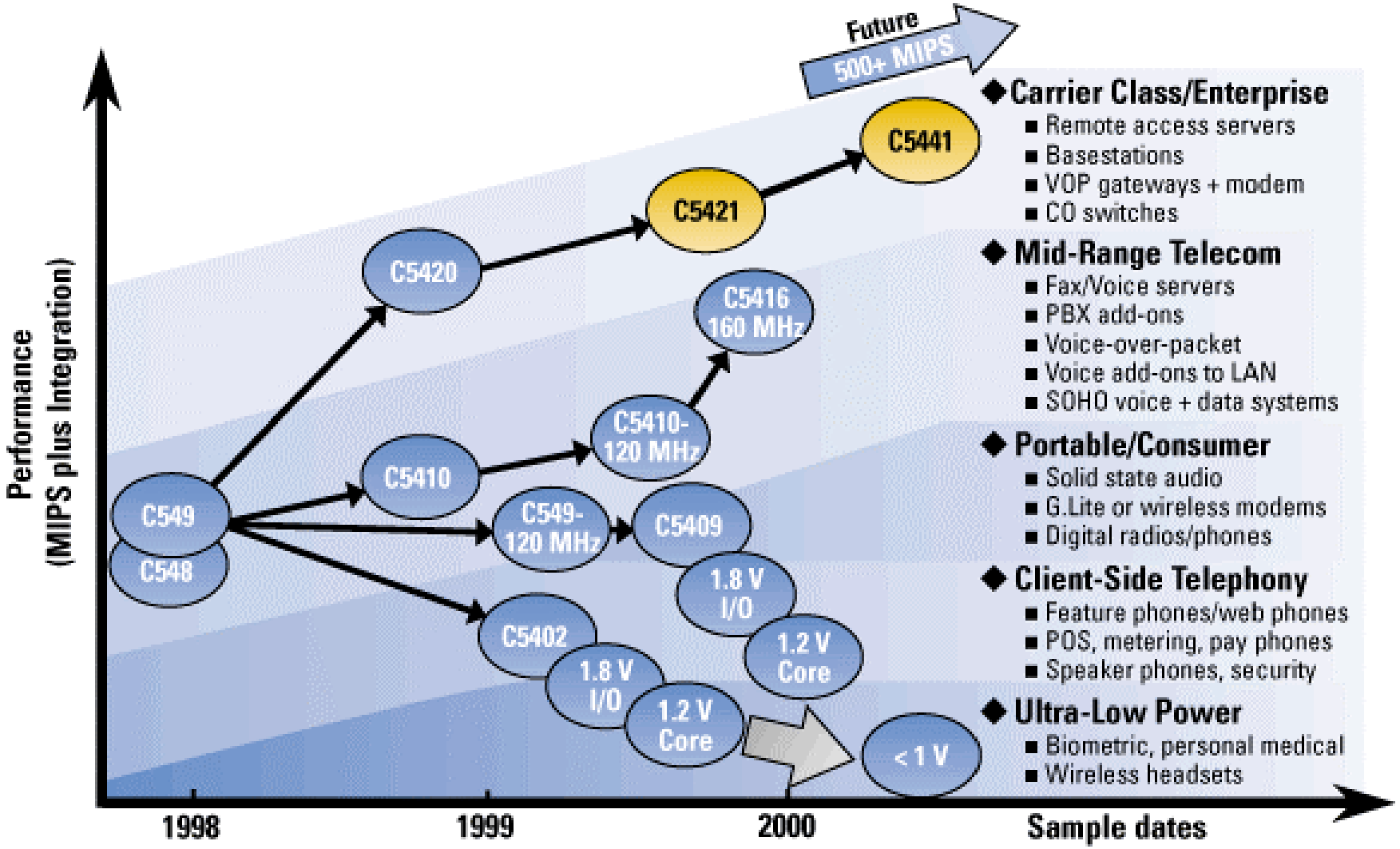
- **Cellular phone - TMS320C540**
- **Fax/ voice server**

Low end

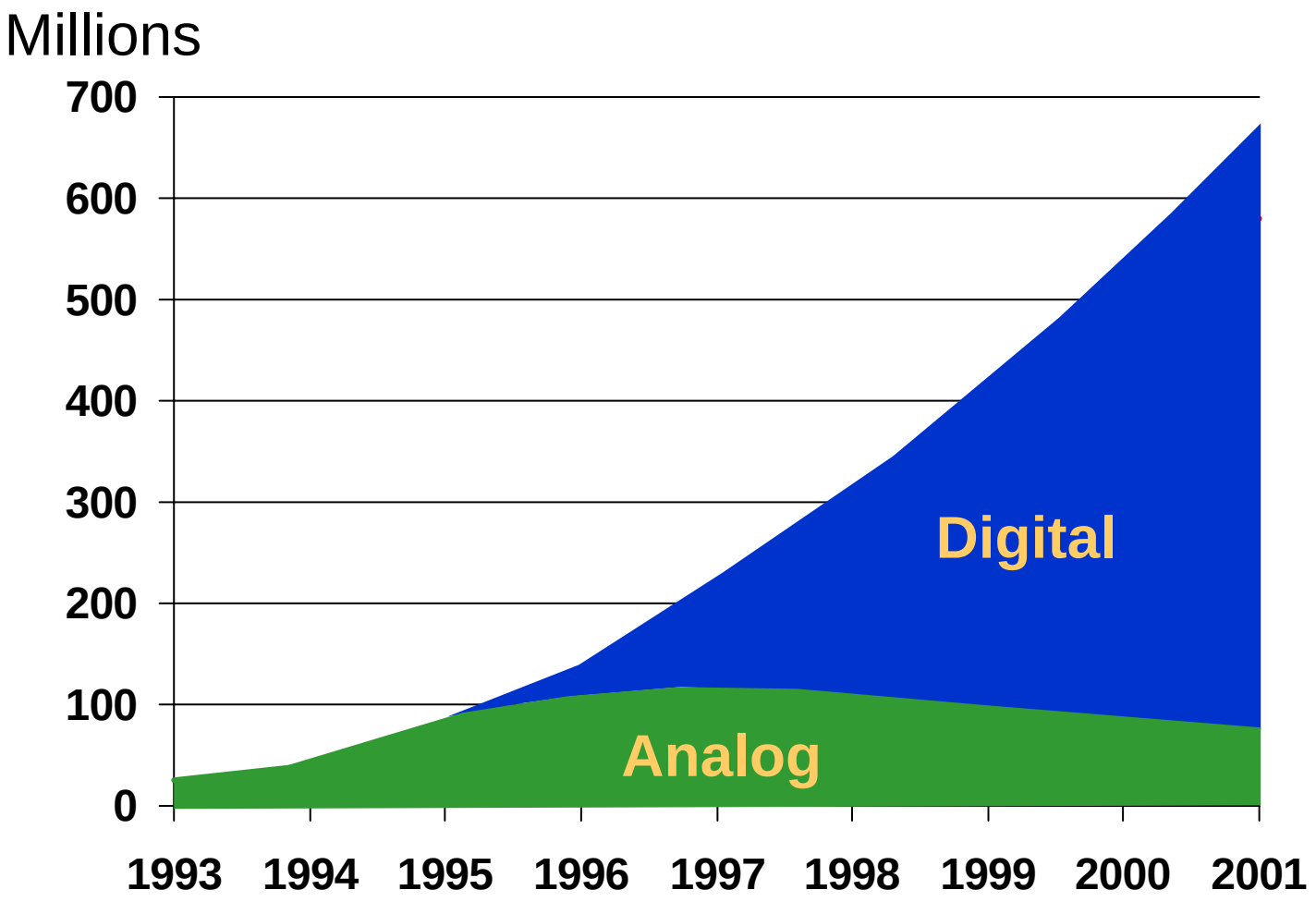
- **Storage products - TMS320C27**
- **Digital camera - TMS320C5000**
- **Portable phones**
- **Wireless headsets**
- **Consumer audio**
- **Automobiles, toasters, thermostats, ...**



Serving a range of applications



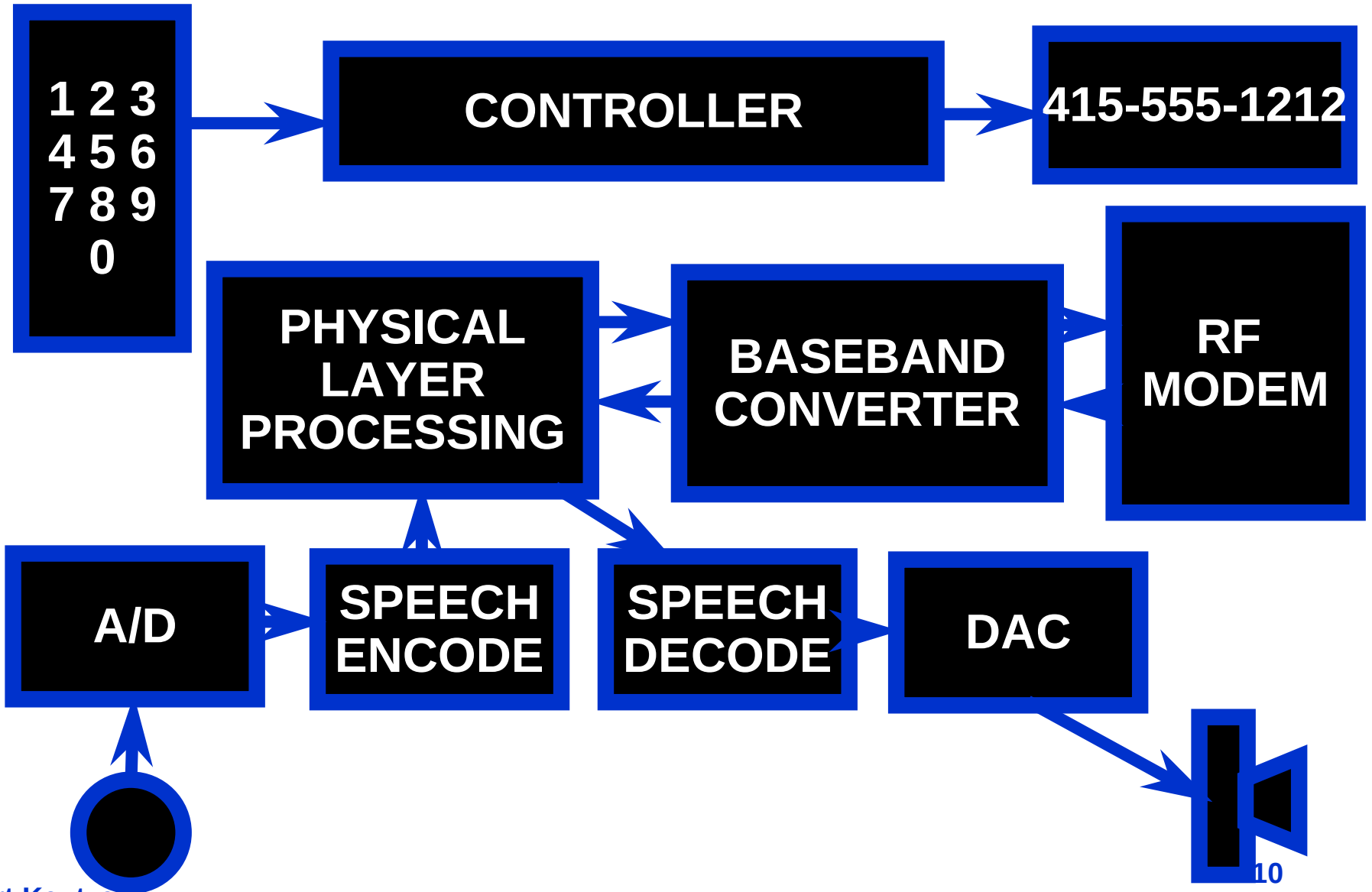
World's Cellular Subscribers



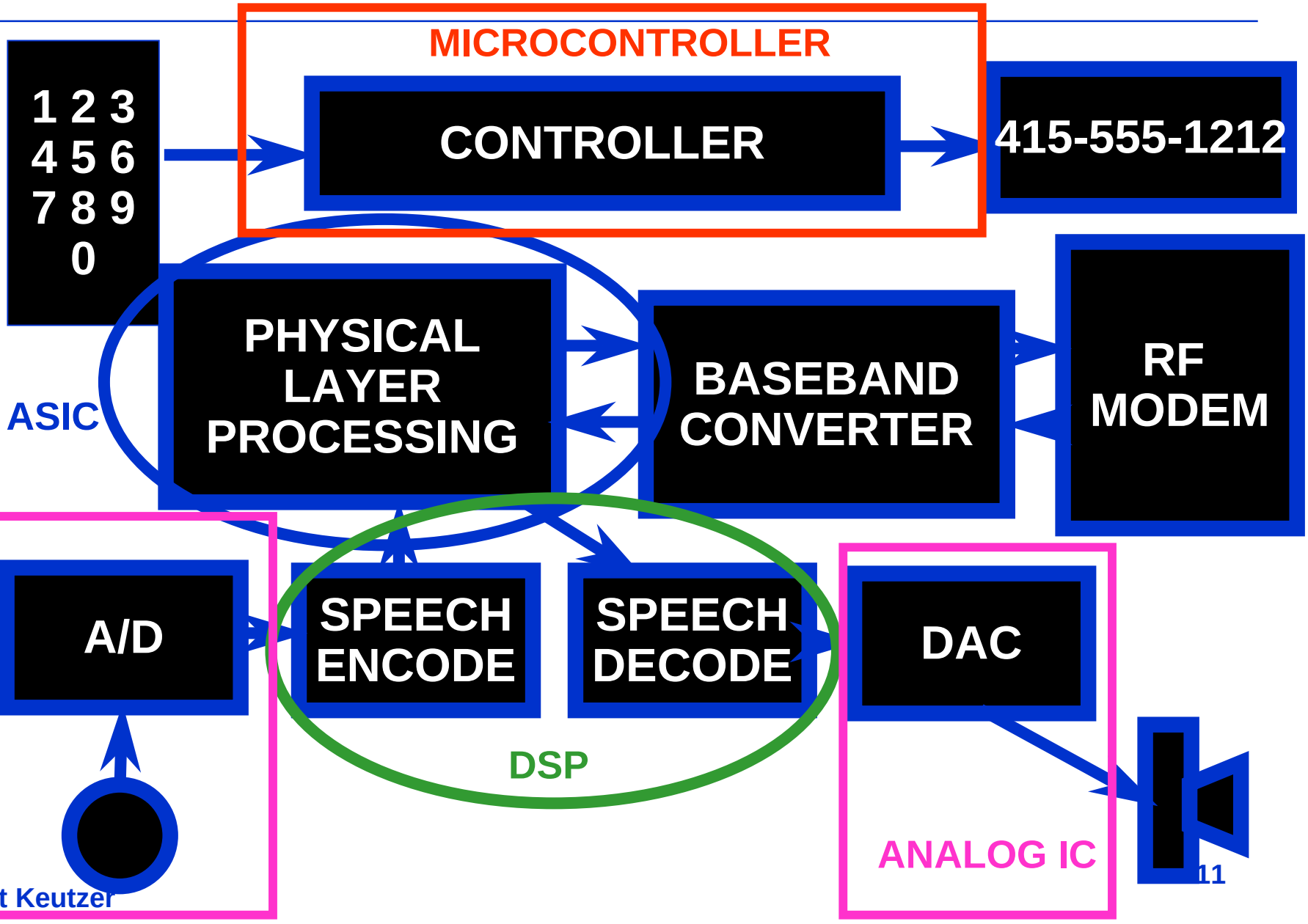
Will provide a ubiquitous infrastructure for wireless data as well as voice

Source: Ericsson Radio Systems, Inc.

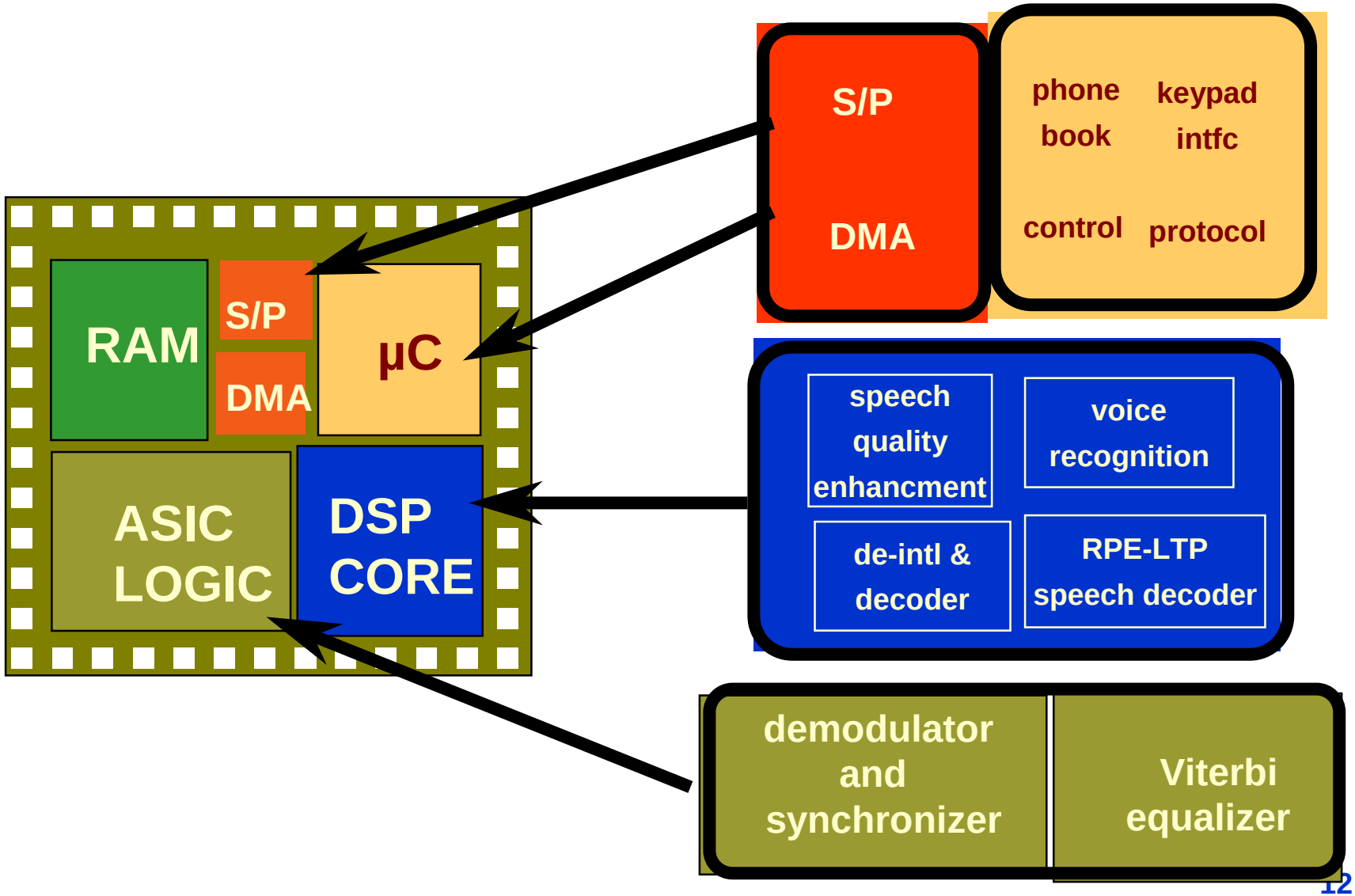
CELLULAR TELEPHONE SYSTEM



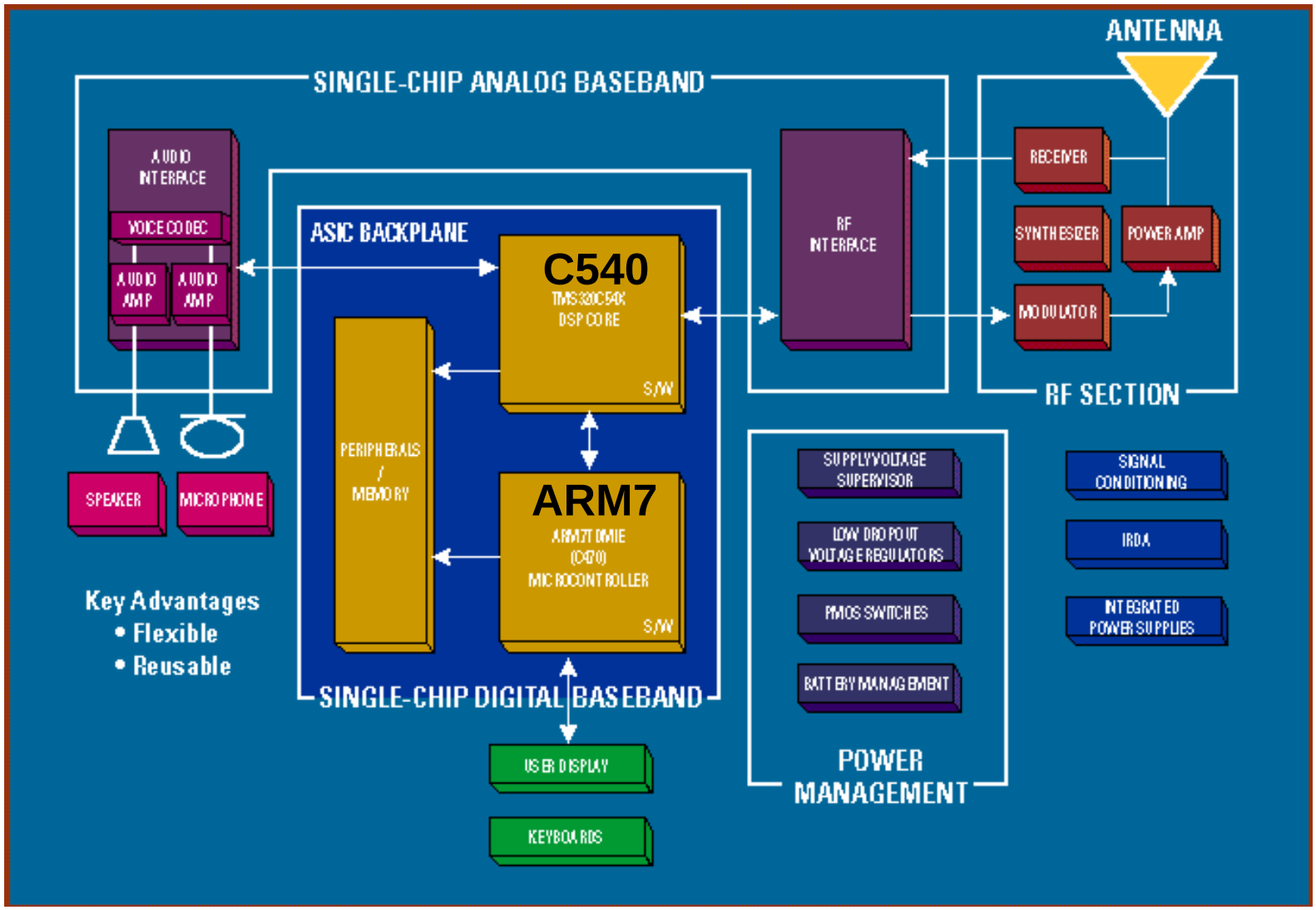
HW/SW/IC PARTITIONING



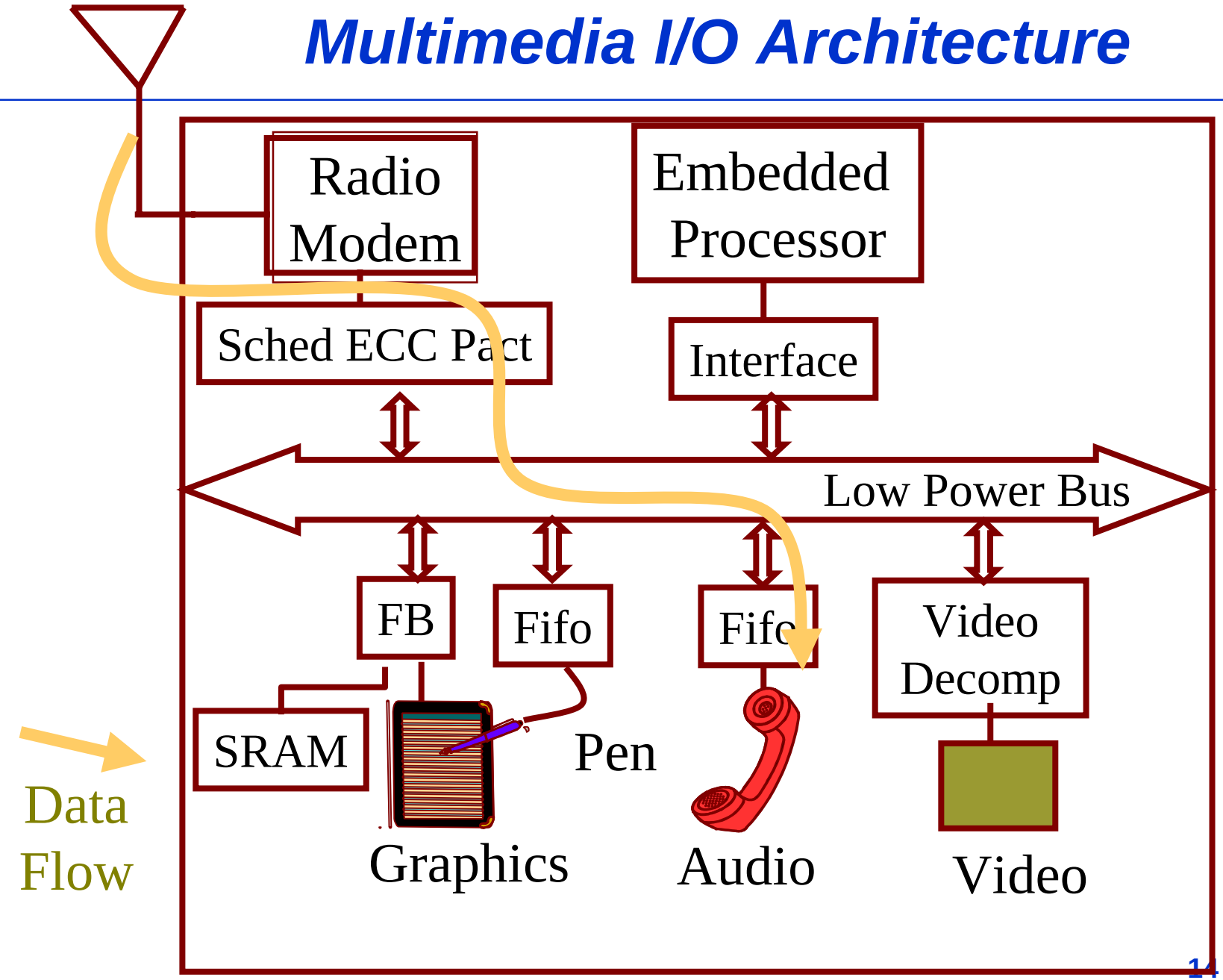
Mapping onto a system on a chip



Example Wireless Phone Organization

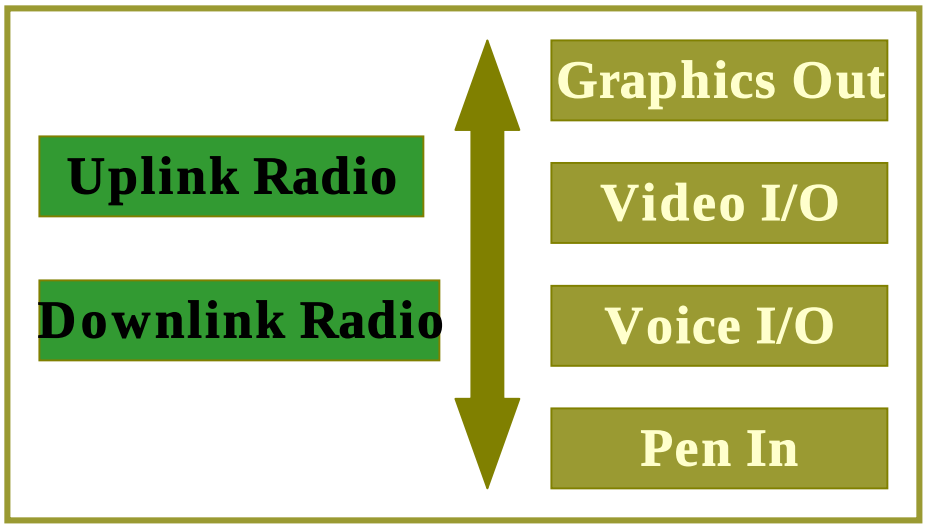


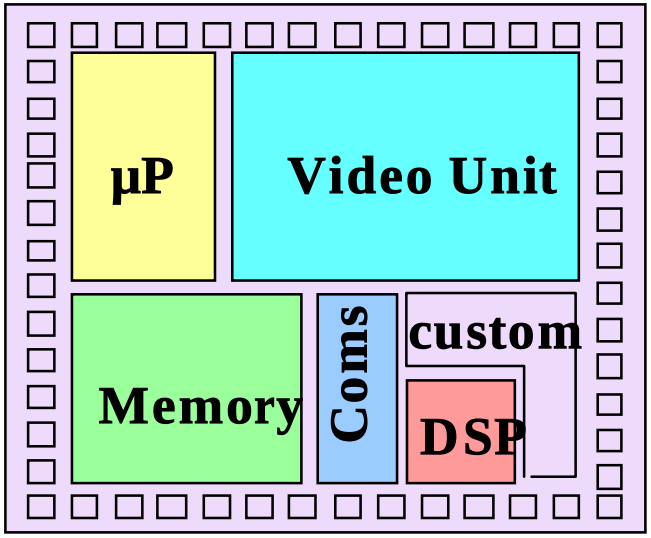
Multimedia I/O Architecture



Multimedia System on a Chip

E.g. Multimedia terminal electronics





Future chips will be a mix of processors, memory and dedicated hardware for specific algorithms and I/O

Requirements of the Embedded Processors

Optimized for a single program - code often in on-chip ROM or off chip EPROM

Minimum code size (one of the motivations initially for Java)

Performance obtained by optimizing datapath

Low cost

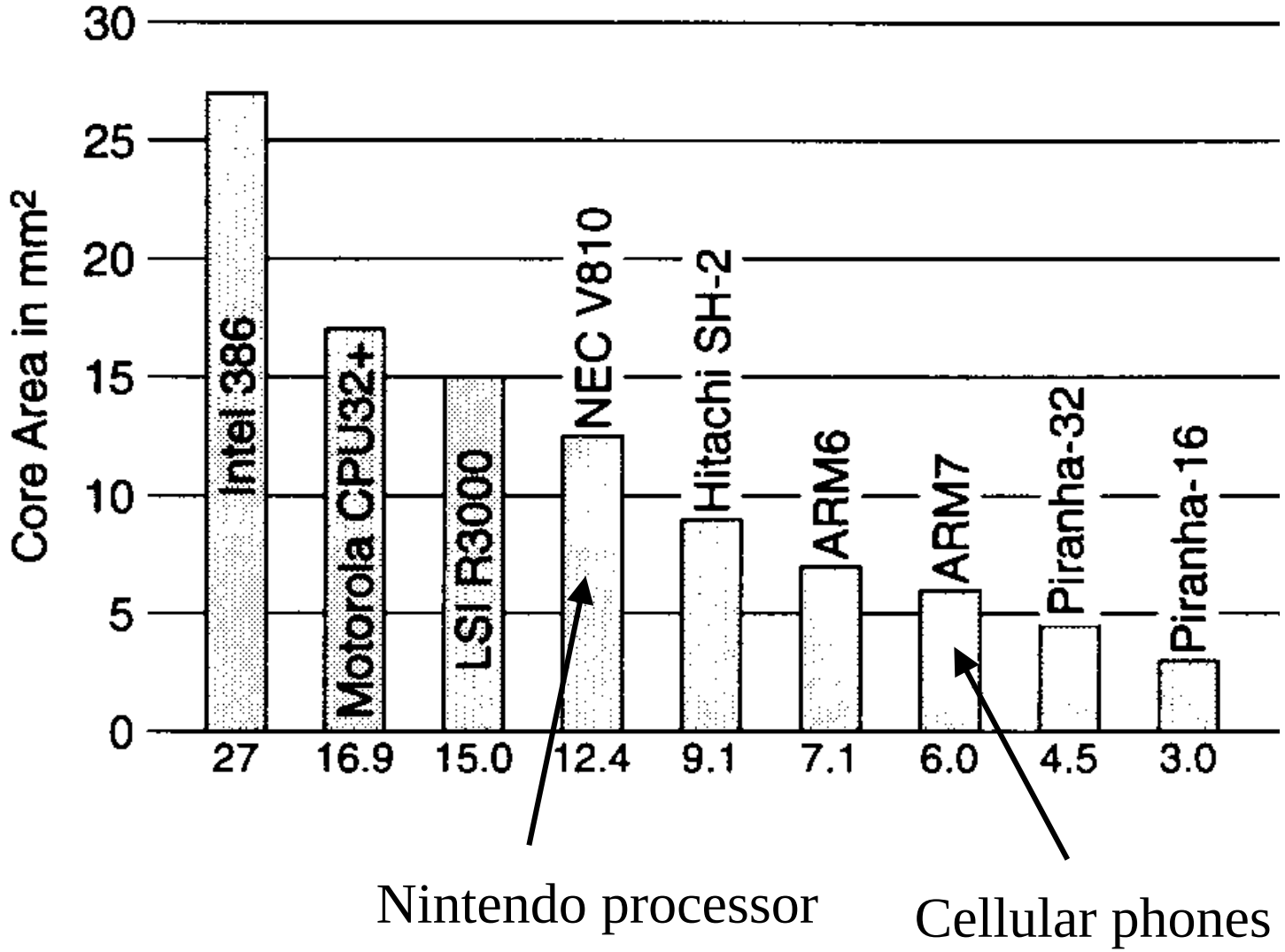
- Lowest possible area
- Technology behind the leading edge
- High level of integration of peripherals (reduces system cost)

Fast time to market

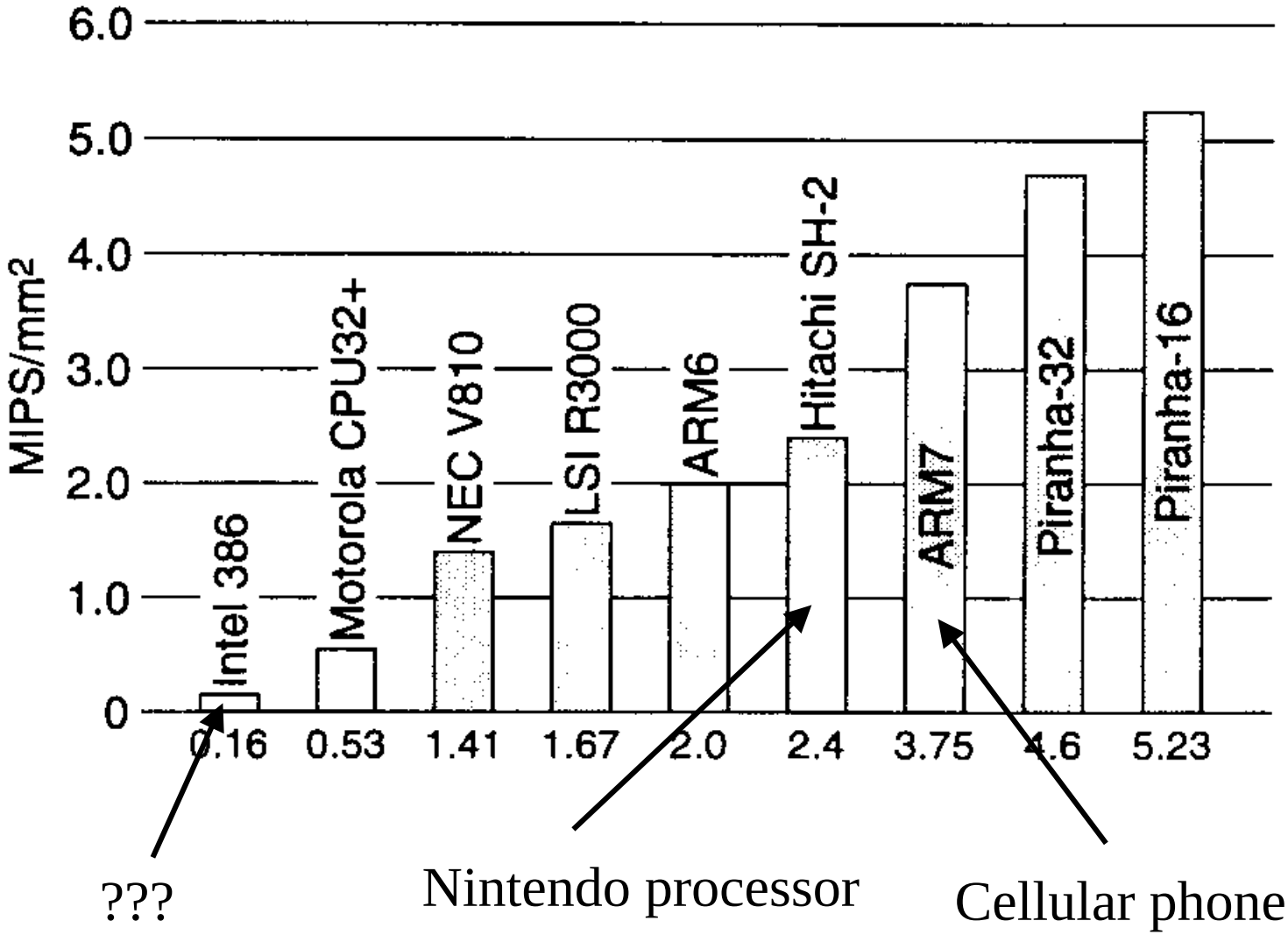
- Compatible architectures (e.g. ARM) allows reuseable code
- Customizable core

Low power if application requires portability

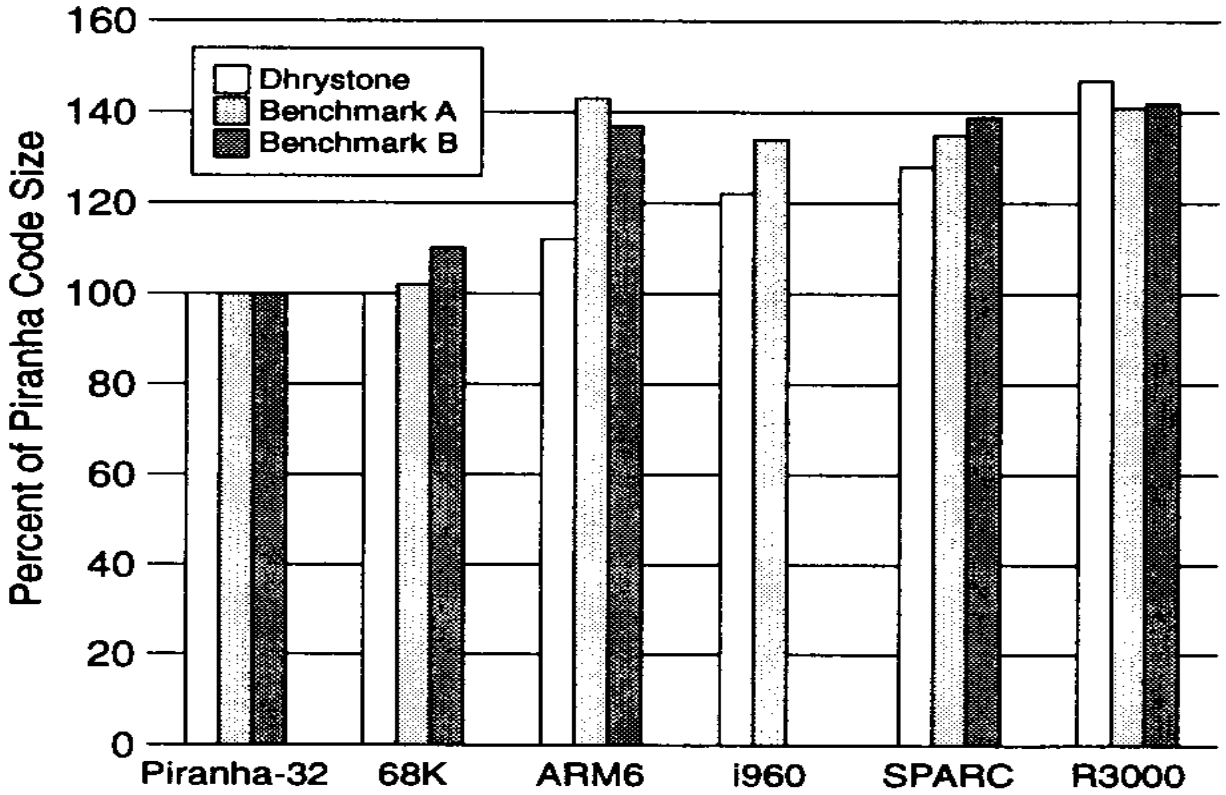
Area of processor cores = Cost



Another figure of merit
Computation per unit area



Code size



If a majority of the chip is the program stored in ROM, then code size is a critical issue

The Piranha has 3 sized instructions - basic 2 byte, and 2 byte plus 16 or 32 bit immediate

BENCHMARKS - DSPstone

ZIVOJNOVIC, VERLADE, SCHLAGER: UNIVERSITY OF AACHEN

APPLICATION BENCHMARKS

- **ADPCM TRANSCODER - CCITT G.721**
- **REAL_UPDATE**
- **COMPLEX_UPDATES**
- **DOT_PRODUCT**
- **MATRIX_1X3**
- **CONVOLUTION**
- **FIR**
- **FIR2DIM**
- **HR_ONE_BIQUAD**
- **LMS**
- **FFT_INPUT_SCALED**

Evolution of GP and DSP

General Purpose Microprocessor traces roots back to Eckert, Mauchly, Von Neumann (ENIAC)

DSP evolved from Analog Signal Processors, using analog hardware to transform phyical signals (classical electrical engineering)

ASP to DSP because

- DSP insensitive to environment (e.g., same response in snow or desert if it works at all)
- DSP performance identical even with variations in components; 2 analog systems behavior varies even if built with same components with 1% variation

Different history and different applications led to different terms, different metrics, some new inventions

Convergence of markets will lead to architectural showdown

Embedded Systems vs. General Purpose Computing - 1

Embedded System	General purpose computing
Runs a few applications often known at design time	Intended to run a fully general set of applications
Not end-user programmable	End-user programmable
Operates in fixed run-time constraints, additional performance may not be useful/valuable	Faster is always better

Embedded Systems vs. General Purpose Computing - 2

Embedded System

Differentiating features:

- **power**
- **cost**
- **speed (must be predictable)**

General purpose computing

Differentiating features

- **speed (need not be fully predictable)**
- **speed**
- **did we mention speed?**
- **cost (largest component power)**

DSP vs. General Purpose MPU

DSPs tend to be written for 1 program, not many programs.

- Hence OSes are much simpler, there is no virtual memory or protection, ...

DSPs sometimes run hard real-time apps

- You must account for anything that could happen in a time slot
- All possible interrupts or exceptions must be accounted for and their collective time be subtracted from the time interval.
- Therefore, exceptions are BAD!

DSPs have an infinite continuous data stream

DSP vs. General Purpose MPU

The “MIPS/MFLOPS” of DSPs is speed of Multiply-Accumulate (MAC).

- DSP are judged by whether they can keep the multipliers busy 100% of the time.

The "SPEC" of DSPs is 4 algorithms:

- Inifinite Impule Response (IIR) filters
- Finite Impule Response (FIR) filters
- FFT, and
- convolvers

In DSPs, algorithms are king!

- Binary compatability not an issue

Software is not (yet) king in DSPs.

- People still write in assembly language for a product to minimize the die area for ROM in the DSP chip.

TYPES OF DSP PROCESSORS

DSP Multiprocessors on a die

- **TMS320C80**
- **TMS320C6000**

32-BIT FLOATING POINT

- **TI TMS320C4X**
- **MOTOROLA 96000**
- **AT&T DSP32C**
- **ANALOG DEVICES ADSP21000**

16-BIT FIXED POINT

- **TI TMS320C2X**
- **MOTOROLA 56000**
- **AT&T DSP16**
- **ANALOG DEVICES ADSP2100**

Note of Caution on DSP Architectures

Successful DSP architectures have two aspects:

- **Key architectural and micro-architectural features that enabled product success in key parameters**
  - **Speed**
  - **Code density**
  - **Low power**
- **Architectural and micro-architectural features that are artifacts of the era in which they were designed**

- ***We will focus on the former*!**

Architectural Features of DSPs

Data path configured for DSP

- **Fixed-point arithmetic**
- **MAC- Multiply-accumulate**

Multiple memory banks and buses -

- **Harvard Architecture**
- **Multiple data memories**

Specialized addressing modes

- **Bit-reversed addressing**
- **Circular buffers**

Specialized instruction set and execution control

- **Zero-overhead loops**
- **Support for MAC**

Specialized peripherals for DSP

THE ULTIMATE IN BENCHMARK DRIVEN ARCHITECTURE DESIGN!!!

DSP Data Path: Arithmetic

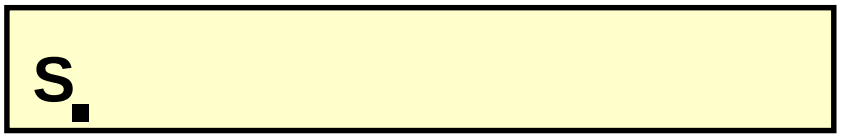
DSPs dealing with numbers representing real world

=> Want “reals”/ fractions

DSPs dealing with numbers for addresses

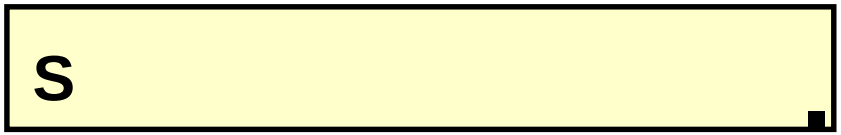
=> Want integers

Support “fixed point” as well as integers



radix point

$-1 \leq x < 1$



radix point

$-2^{N-1} \leq x < 2^{N-1}$

DSP Data Path: Precision

Word size affects precision of fixed point numbers

DSPs have 16-bit, 20-bit, or 24-bit data words

Floating Point DSPs cost 2X - 4X vs. fixed point, slower than fixed point

DSP programmers will scale values inside code

- SW Libraries
- Separate explicit exponent

"Blocked Floating Point" single exponent for a group of fractions

Floating point support simplify development

DSP Data Path: Overflow?

DSP are descended from analog :

what should happen to output when “peg” an input?

(e.g., turn up volume control knob on stereo)

- Modulo Arithmetic???

Set to most positive ($2^{N-1}-1$) or

most negative value(-2^{N-1}) : “saturation”

Many algorithms were developed in this model

DSP Data Path: Multiplier

Specialized hardware performs all key arithmetic operations in 1 cycle

50% of instructions can involve multiplier
=> single cycle latency multiplier

Need to perform multiply-accumulate (MAC)

n-bit multiplier => 2n-bit product

DSP Data Path: Accumulator

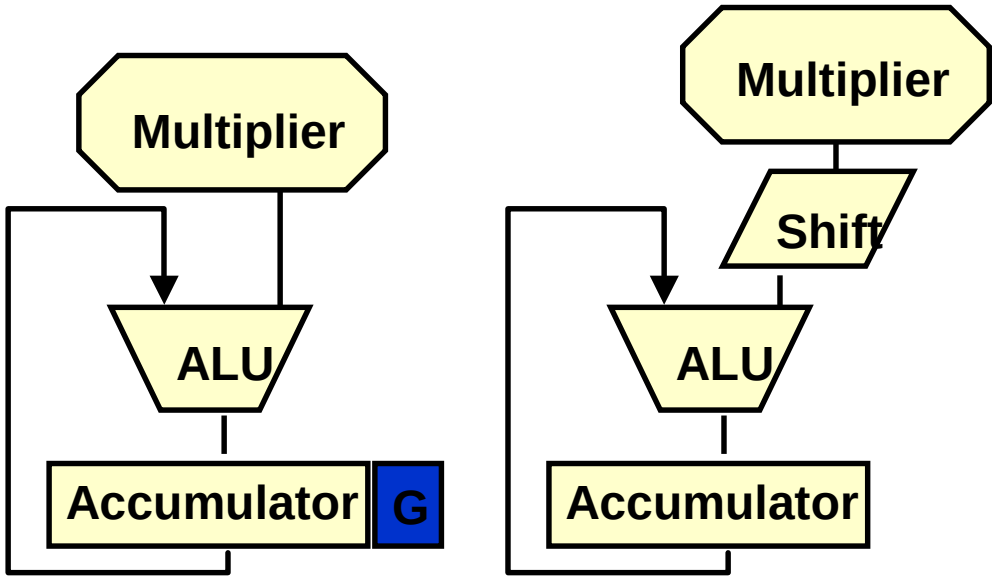
Don’t want overflow or have to scale accumulator

Option 1: accumalator wider than product:

“guard bits”

- Motorola DSP:
  24b x 24b => 48b product, 56b Accumulator

Option 2: shift right and round product before adder



DSP Data Path: Rounding

Even with guard bits, will need to round when store accumulator into memory

3 DSP standard options

Truncation: chop results

=> biases results up

Round to nearest:

< 1/2 round down, 1/2 round up (more positive)

=> smaller bias

Convergent:

< 1/2 round down, > 1/2 round up (more positive), = 1/2 round to make lsb a zero (+1 if 1, +0 if 0)

=> no bias

IEEE 754 calls this round to nearest even

Data Path

DSP Processor

Specialized hardware performs all key arithmetic operations in 1 cycle.

Hardware support for managing numeric fidelity:

- **Shifters**
- **Guard bits**
- **Saturation**

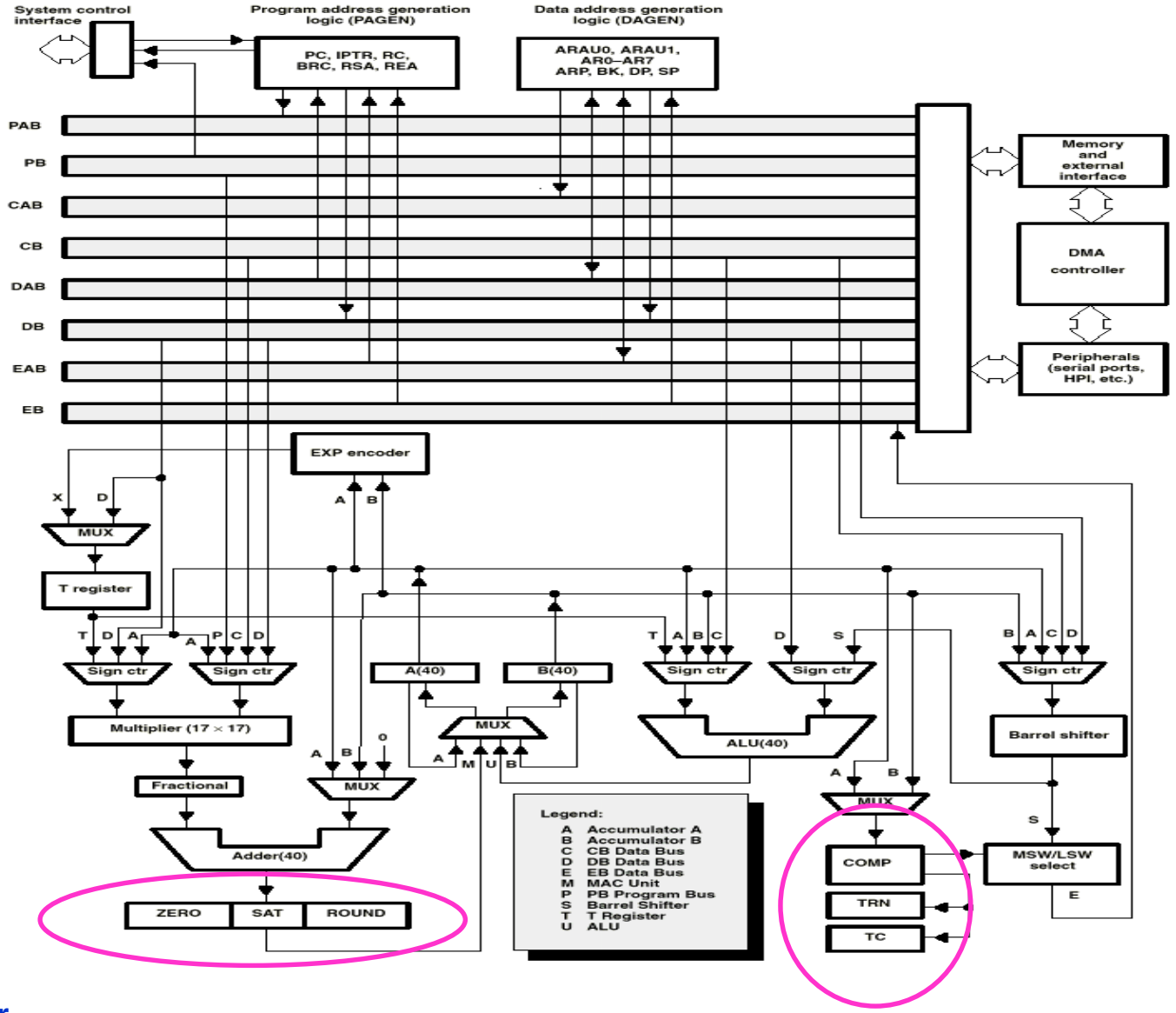
General-Purpose Processor

Multiplies often take>1 cycle

Shifts often take >1 cycle

Other operations (e.g., saturation, rounding) typically take multiple cycles.

320C54x DSP Functional Block Diagram



FIR Filtering: A Motivating Problem

M most recent samples in the delay line (Xi)

New sample moves data down delay line

“Tap” is a multiply-add

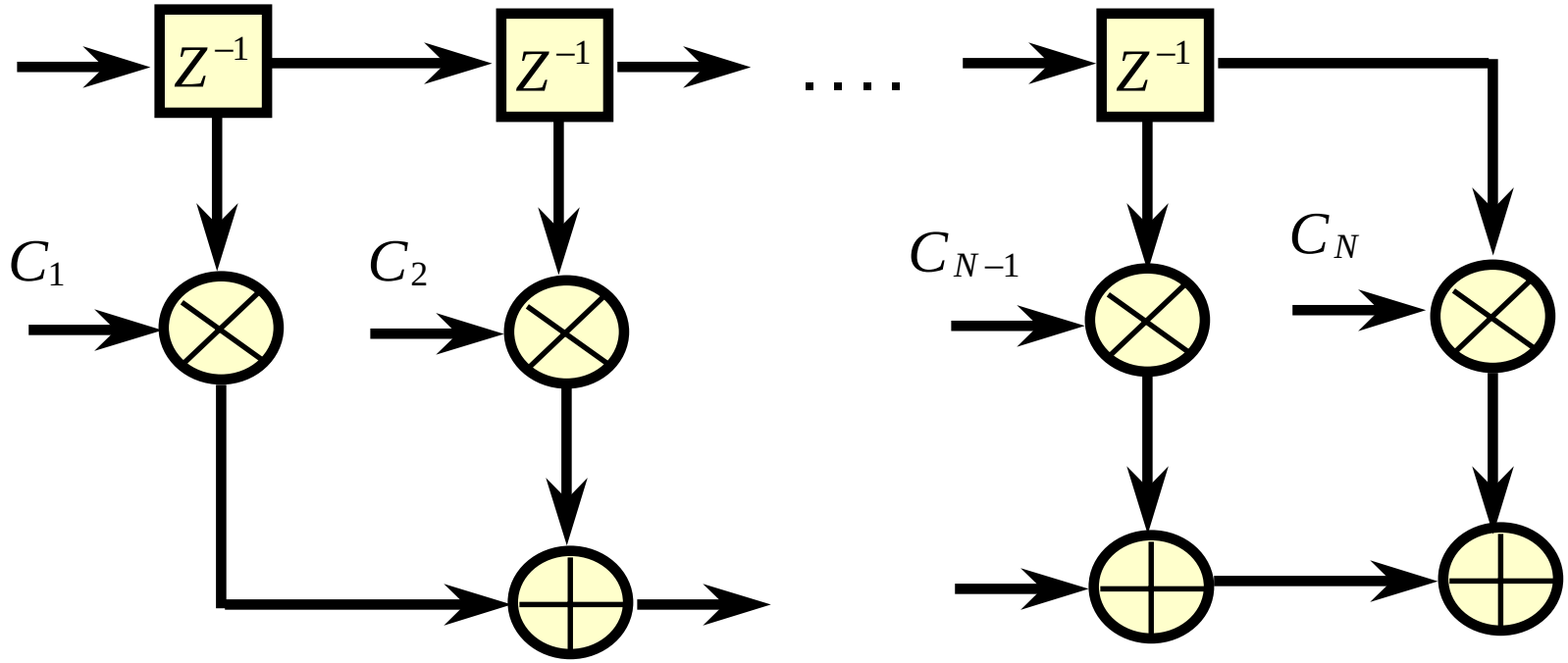
Each tap (M+1 taps total) nominally requires:

- Two data fetches
- Multiply
- Accumulate
- Memory write-back to update delay line

Goal: 1 FIR Tap / DSP instruction cycle

BENCHMARKS - FIR FILTER

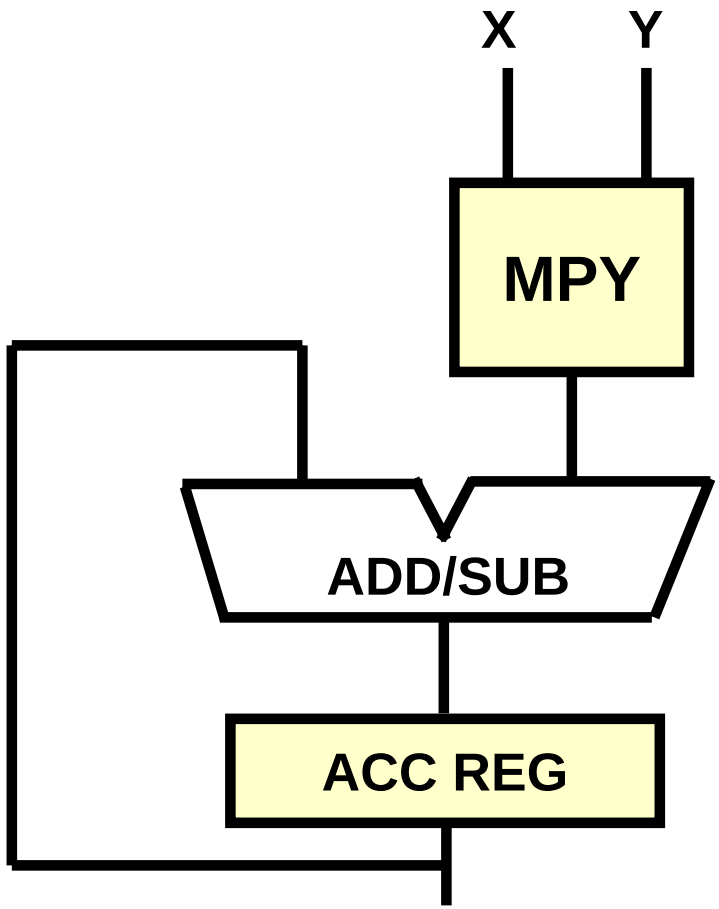
FINITE-IMPULSE RESPONSE FILTER



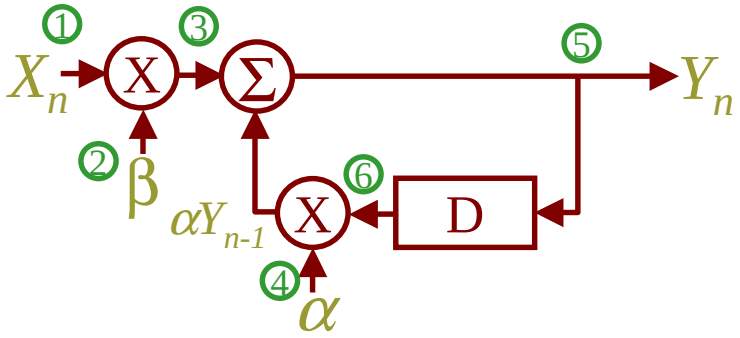
Micro-architectural impact - MAC

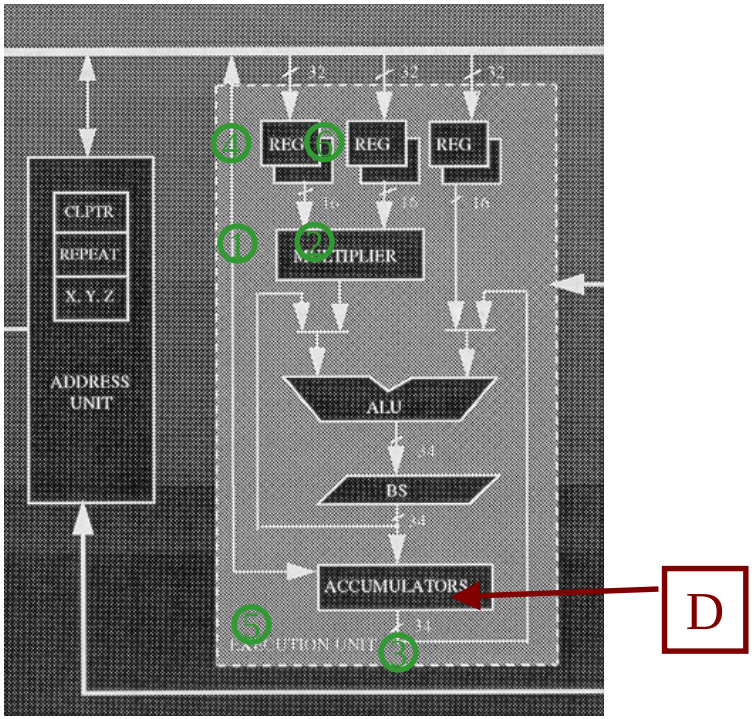
$$y(n) = \sum_{0}^{N-1} h(m)x(n-m)$$

element of finite-impulse response filter computation



Mapping of the filter onto a DSP execution unit

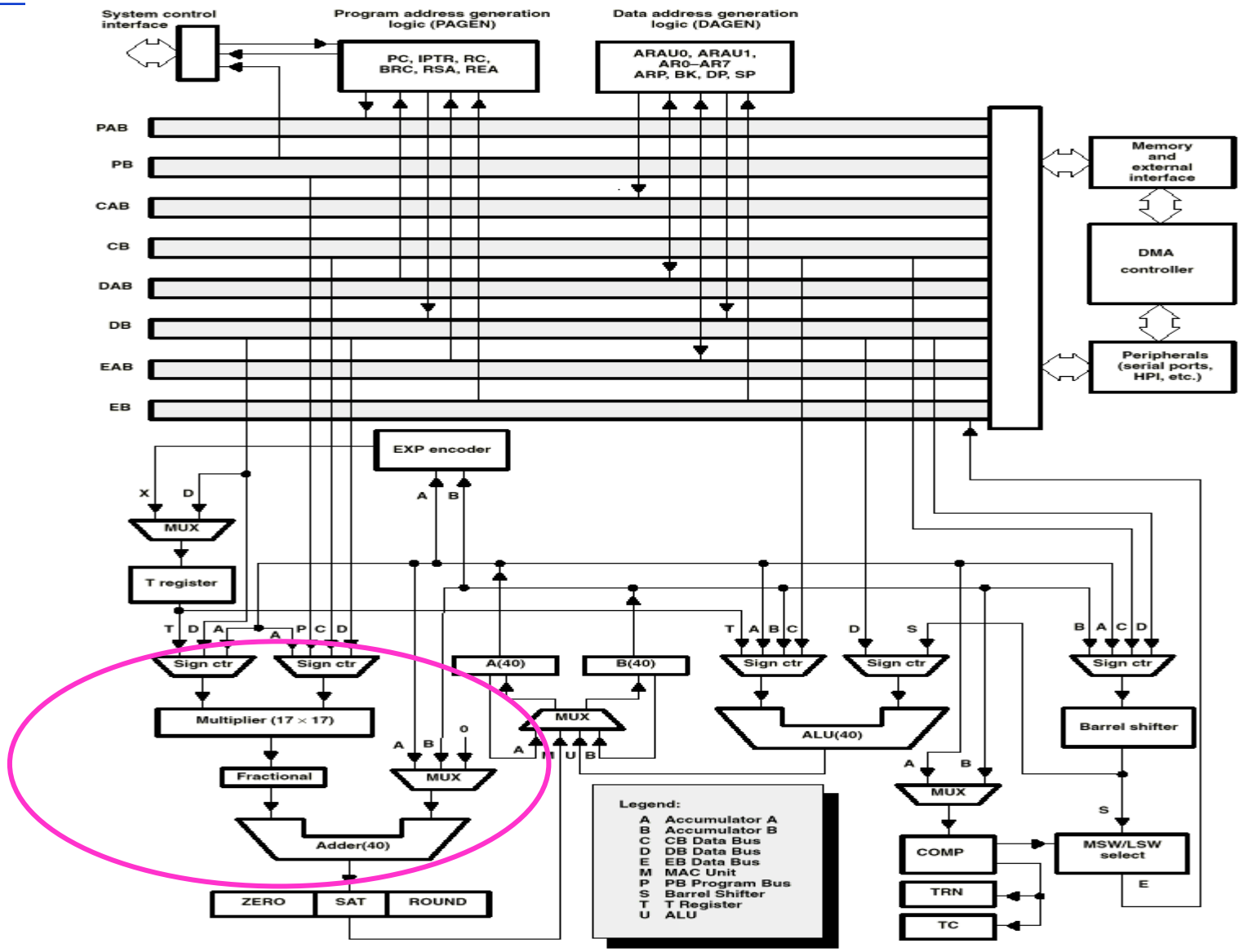




The critical hardware unit in a DSP is the multiplier - much of the architecture is organized around allowing use of the multiplier on every cycle

This means providing two operands on every cycle, through multiple data and address busses, multiple address units and local accumulator feedback

MAC Eg. - 320C54x DSP Functional Block Diagram



DSP Memory

FIR Tap implies multiple memory accesses

DSPs want multiple data ports

Some DSPs have ad hoc techniques to reduce memory bandwdith demand

- Instruction repeat buffer: do 1 instruction 256 times
- Often disables interrupts, thereby increasing interrupt response time

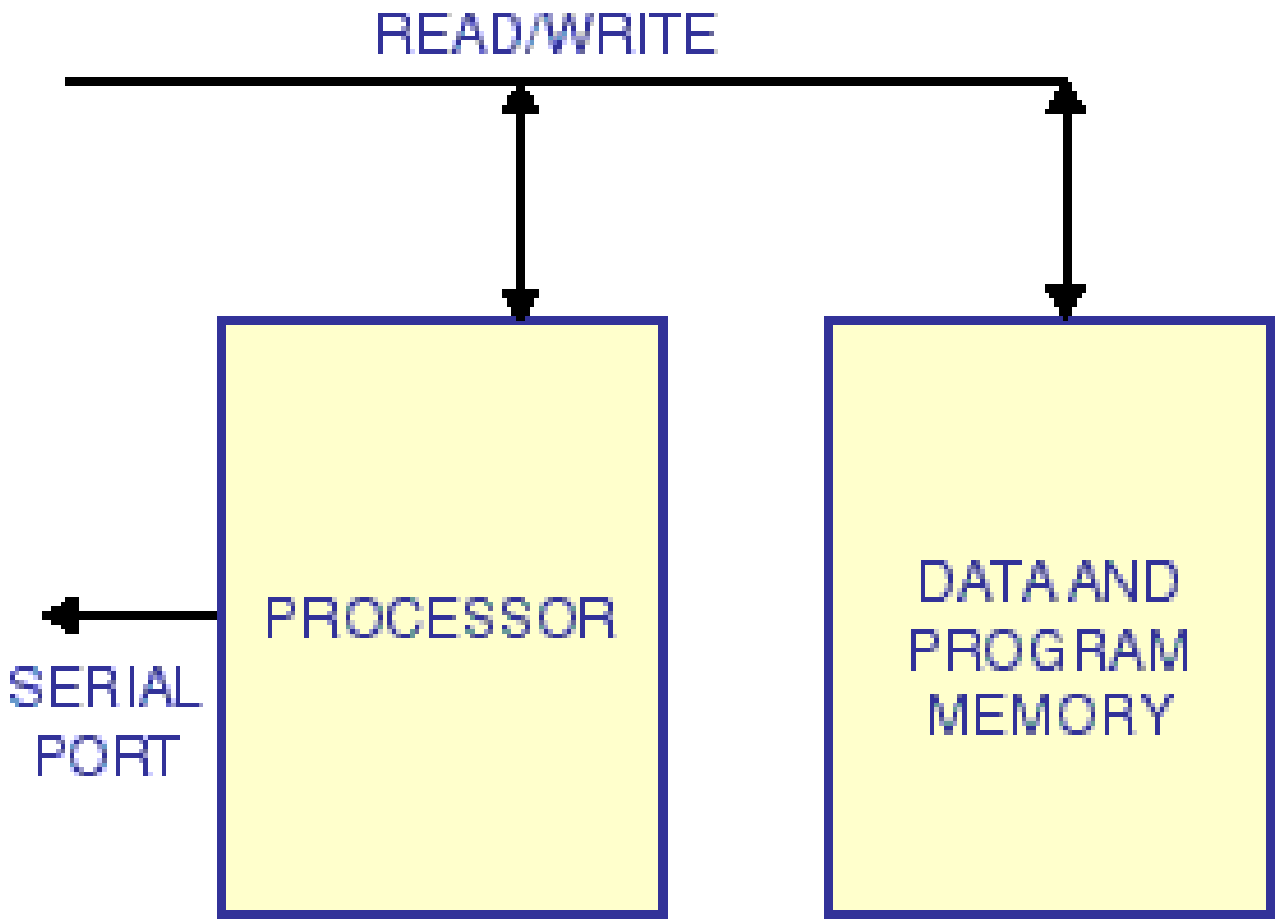
Some recent DSPs have instruction caches

- Even then may allow programmer to “lock in” instructions into cache
- Option to turn cache into fast program memory

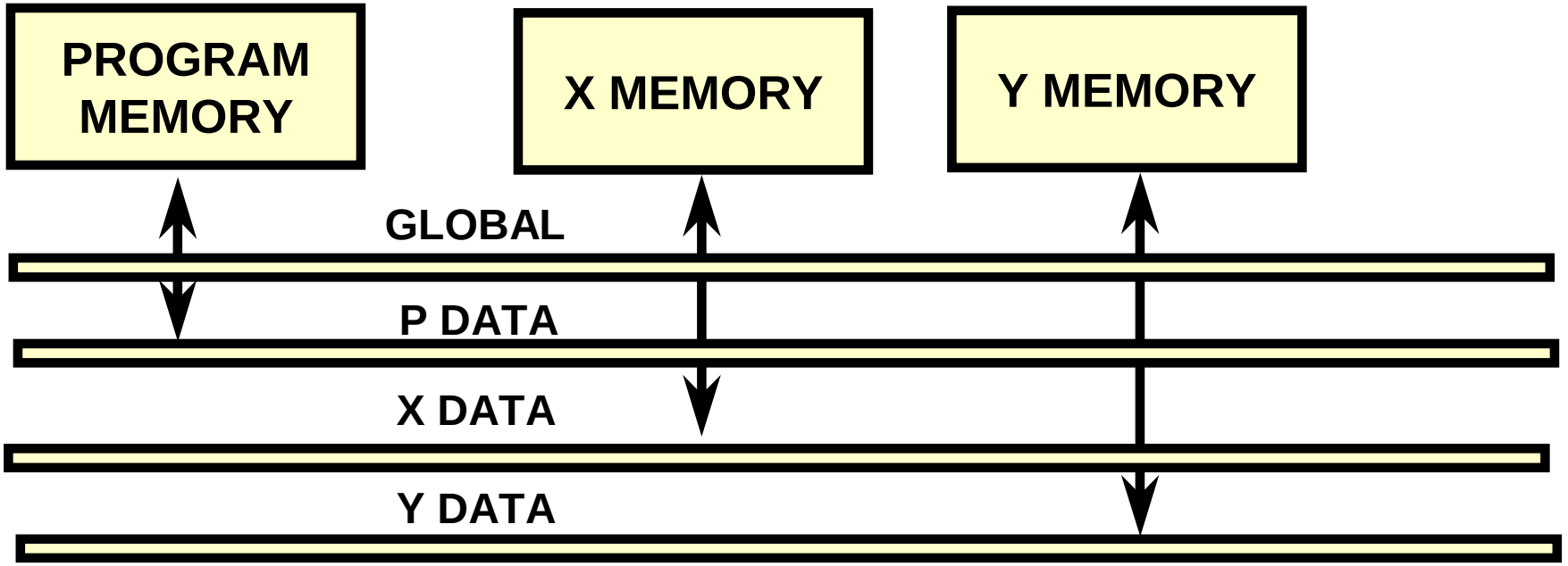
No DSPs have data caches

May have multiple data memories

Conventional ``Von Neumann'' memory



HARVARD ARCHITECTURE in DSP



Memory Architecture

DSP Processor

Harvard architecture

2-4 memory accesses/cycle

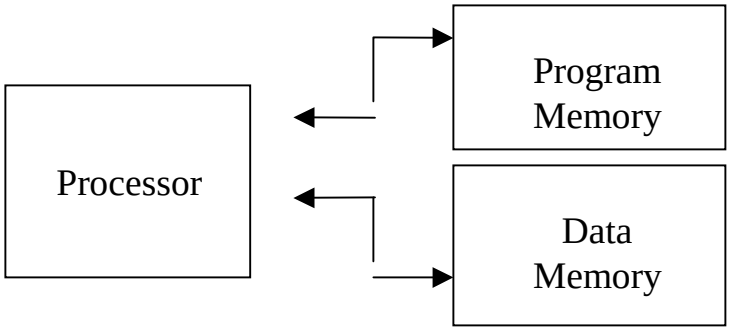
No caches-on-chip SRAM

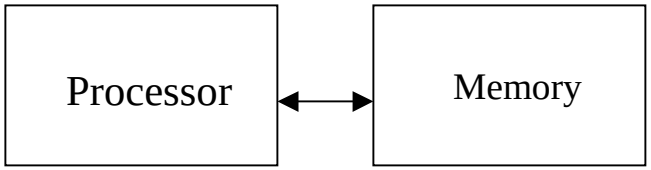
General-Purpose Processor

Von Neumann architecture

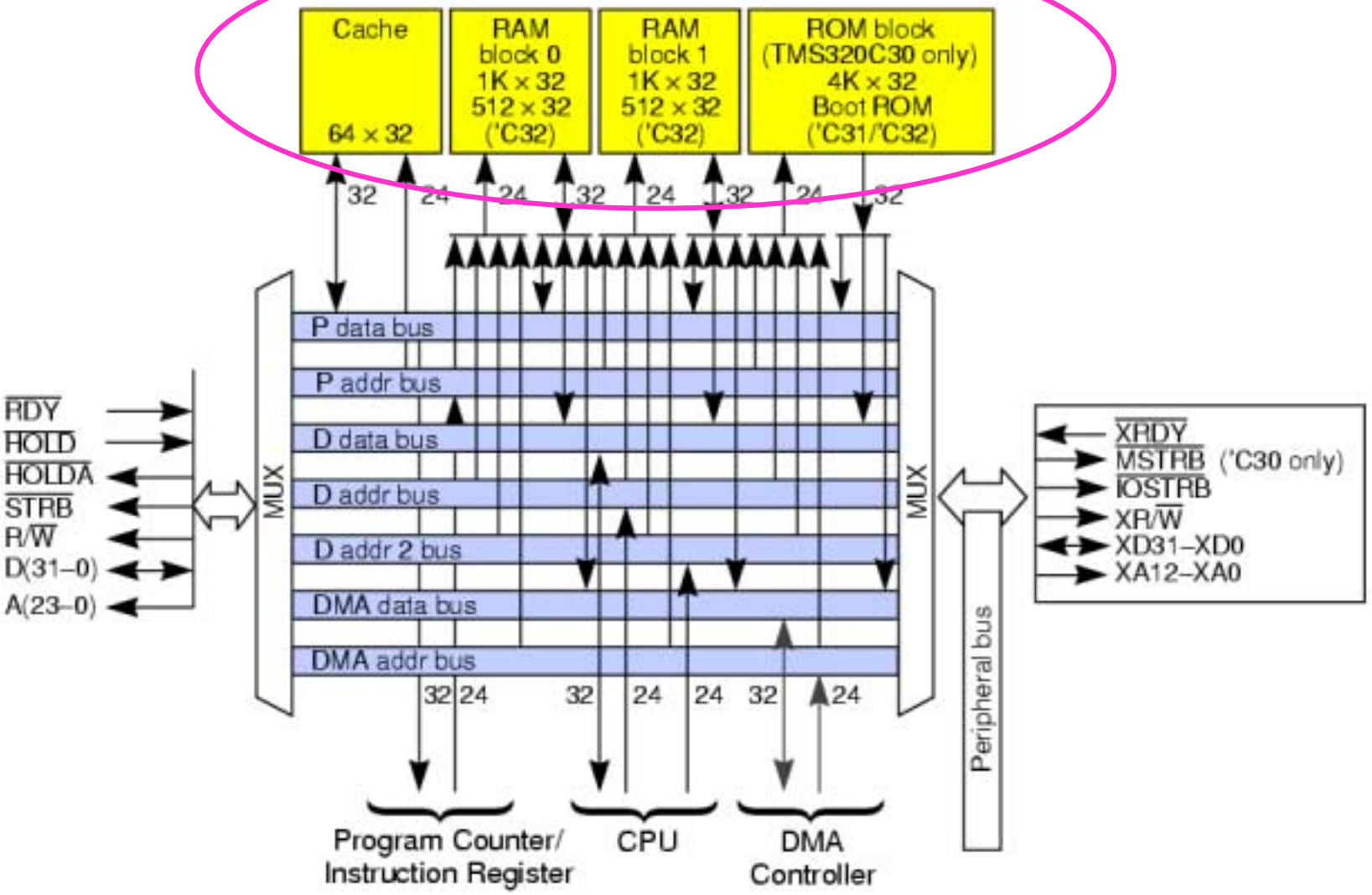
Typically 1 access/cycle

May use caches

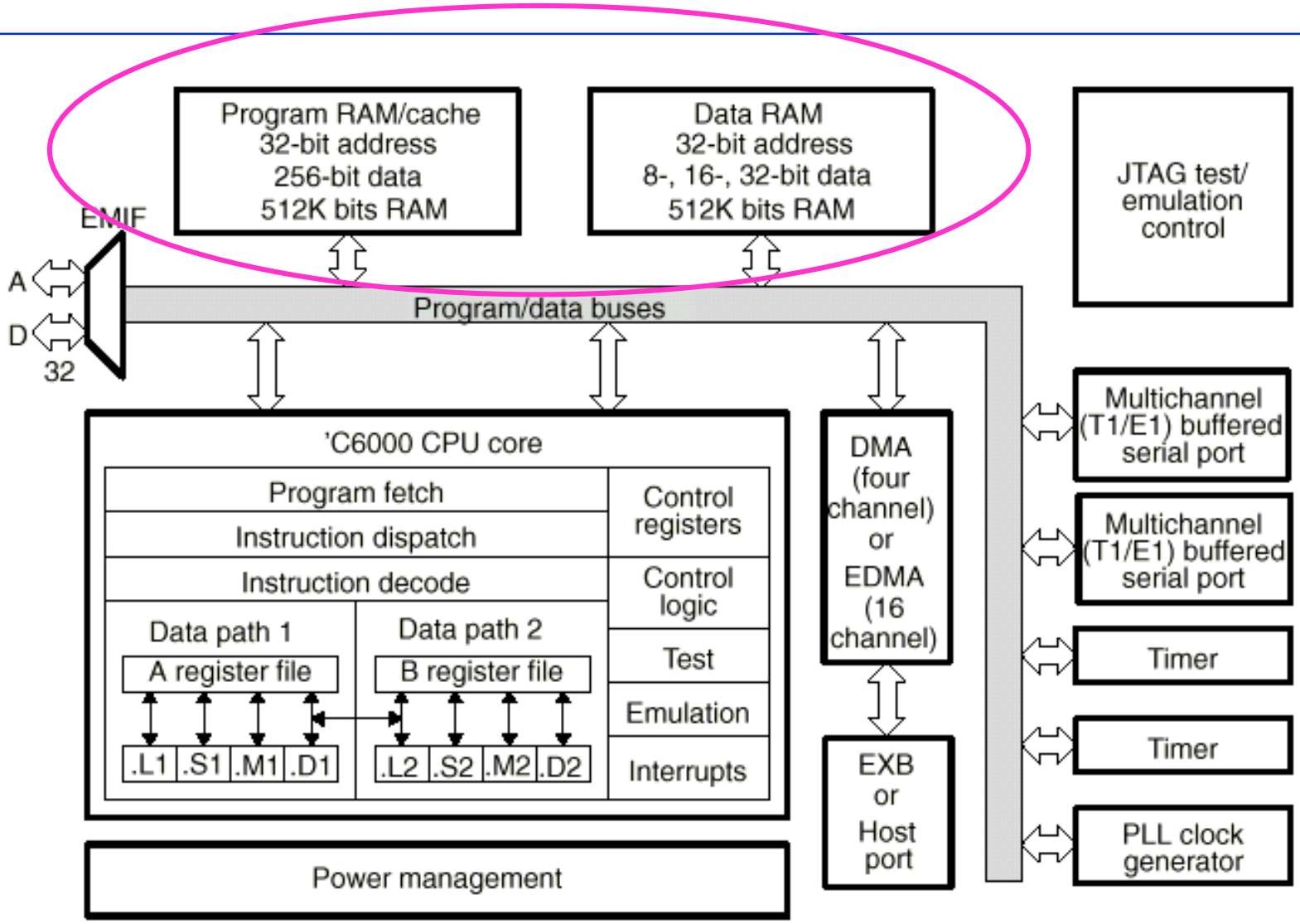




Eg. TMS320C3x MEMORY BLOCK DIAGRAM - Harvard Architecture



Eg. 320C62x/67x DSP



DSP Addressing

Have standard addressing modes: immediate, displacement, register indirect

Want to keep MAC datapth busy

Assumption: any extra instructions imply clock cycles of overhead in inner loop

=> complex addressing is good

=> don't use datapath to calculate fancy address

Autoincrement/Autodecrement register indirect

- lw r1,0(r2)+ => r1 <- M[r2]; r2<-r2+1
- Option to do it before addressing, positive or negative

DSP Addressing: FFT

FFTs start or end with data in weird bufferfly order

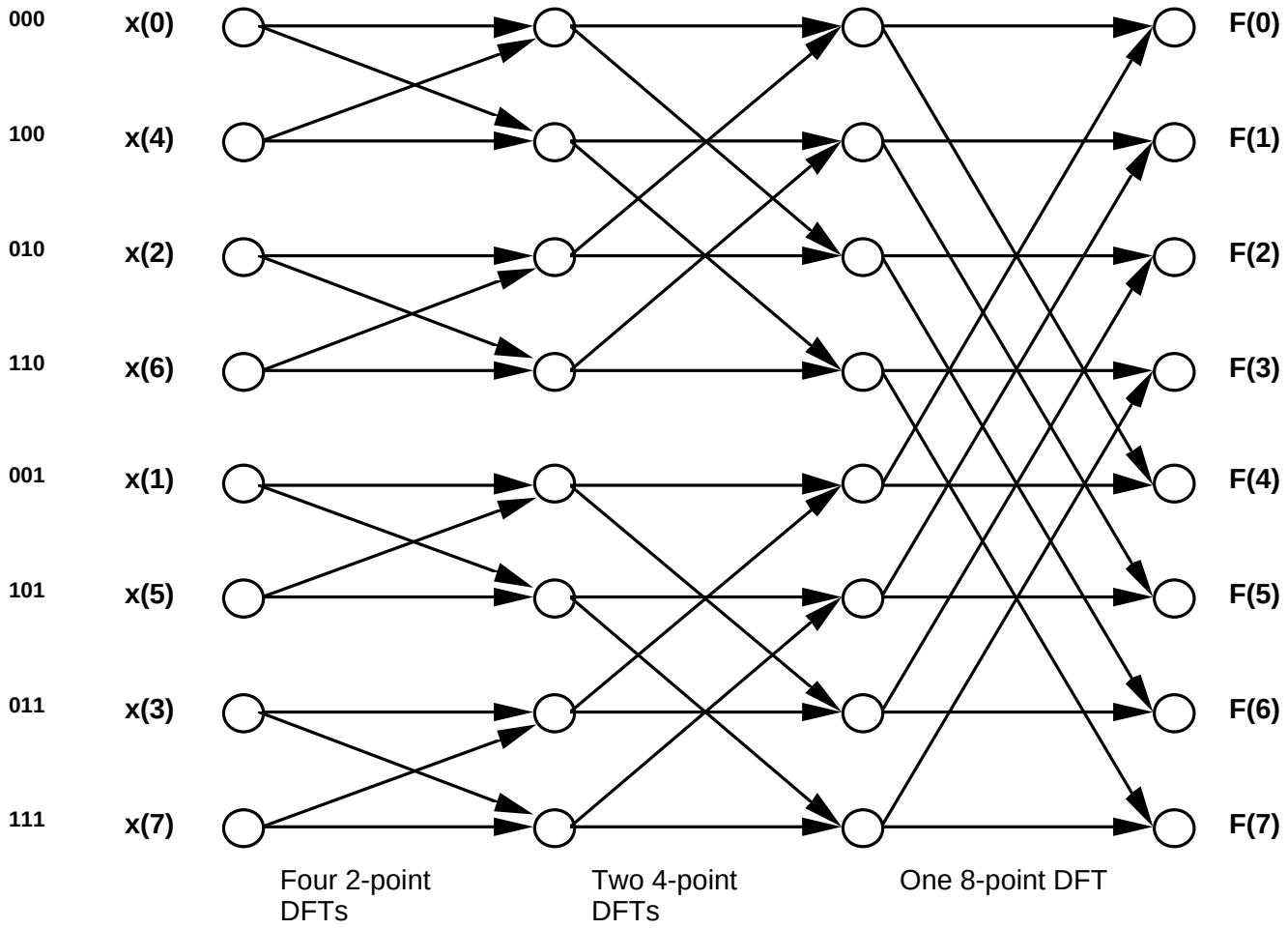
0 (000)	=>	0 (000)
1 (001)	=>	4 (100)
2 (010)	=>	2 (010)
3 (011)	=>	6 (110)
4 (100)	=>	1 (001)
5 (101)	=>	5 (101)
6 (110)	=>	3 (011)
7 (111)	=>	7 (111)

What can do to avoid overhead of address checking instructions for FFT?

Have an optional "bit reverse" address addressing mode for use with autoincrement addressing

Many DSPs have "bit reverse" addressing for radix-2 FFT

BIT REVERSED ADDRESSING



Data flow in the radix-2 decimation-in-time FFT algorithm

DSP Addressing: Buffers

DSPs dealing with continuous I/O

Often interact with an I/O buffer (delay lines)

To save memory, buffer often organized as circular buffer

What can do to avoid overhead of address checking instructions for circular buffer?

Option 1: Keep start register and end register per address register for use with autoincrement addressing, reset to start when reach end of buffer

Option 2: Keep a buffer length register, assuming buffers starts on aligned address, reset to start when reach end

Every DSP has “modulo” or “circular” addressing

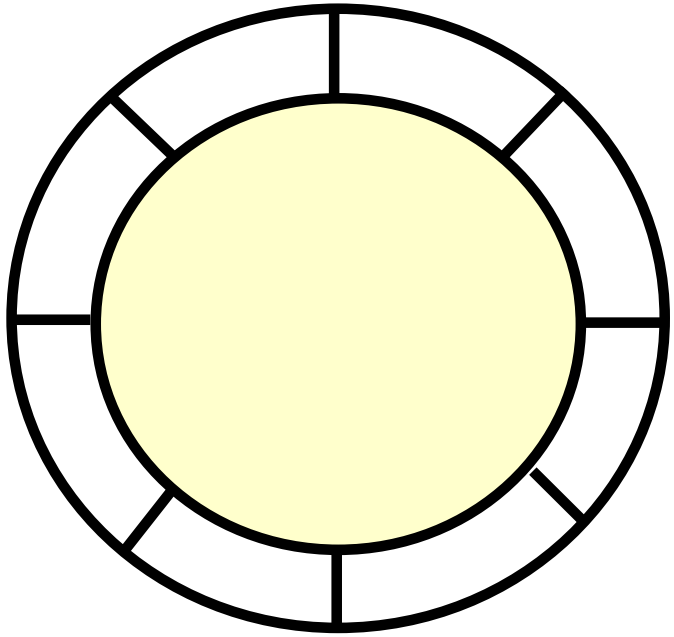
CIRCULAR BUFFERS

Instructions accomodate three elements:

- **buffer address**
- **buffer size**
- **increment**

Allows for cyling through:

- **delay elements**
- **coefficients in data memory**



Addressing

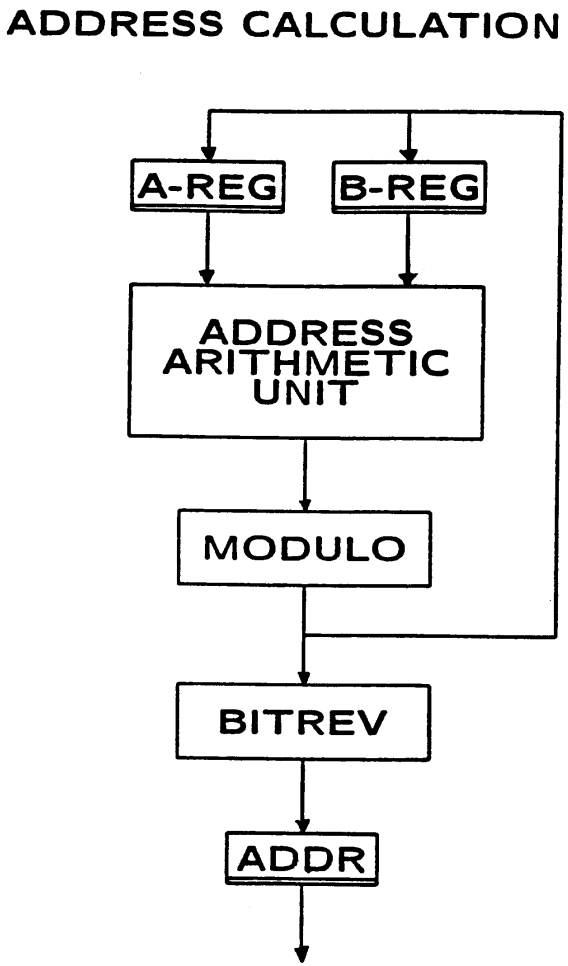
DSP Processor

- Dedicated address generation units
- Specialized addressing modes; e.g.:
  - Autoincrement
  - Modulo (circular)
  - Bit-reversed (for FFT)
- Good immediate data support

General-Purpose Processor

- Often, no separate address generation unit
- General-purpose addressing modes

Address calculation unit for DSP



Supports modulo and bit reversal arithmetic

Often duplicated to calculate multiple addresses per cycle

DSP Instructions and Execution

May specify multiple operations in a single instruction

Must support Multiply-Accumulate (MAC)

Need parallel move support

Usually have special loop support to reduce branch overhead

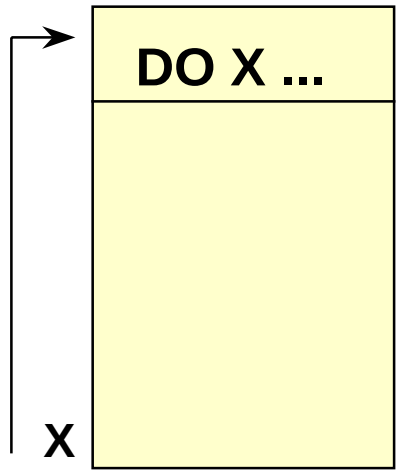
- Loop an instruction or sequence
- 0 value in register usually means loop maximum number of times
- Must be sure if calculate loop count that 0 does not mean 0

May have saturating shift left arithmetic

May have conditional execution to reduce branches

ADSP 2100: ZERO-OVERHEAD LOOP

DO <addr> UNTIL condition”



Address Generation

```
PCS = PC + 1
if (PC = x && ! condition)
   PC = PCS
else
   PC = PC +1
```

- **Eliminates a few instructions in loops -**
- **Important in loops with small bodies**

Instruction Set

DSP Processor

Specialized, complex instructions

Multiple operations per instruction

```
mac x0,y0,a    x: (r0) + ,x0    y: (r4) + ,y0
```

General-Purpose Processor

General-purpose instructions

Typically only one operation per instruction

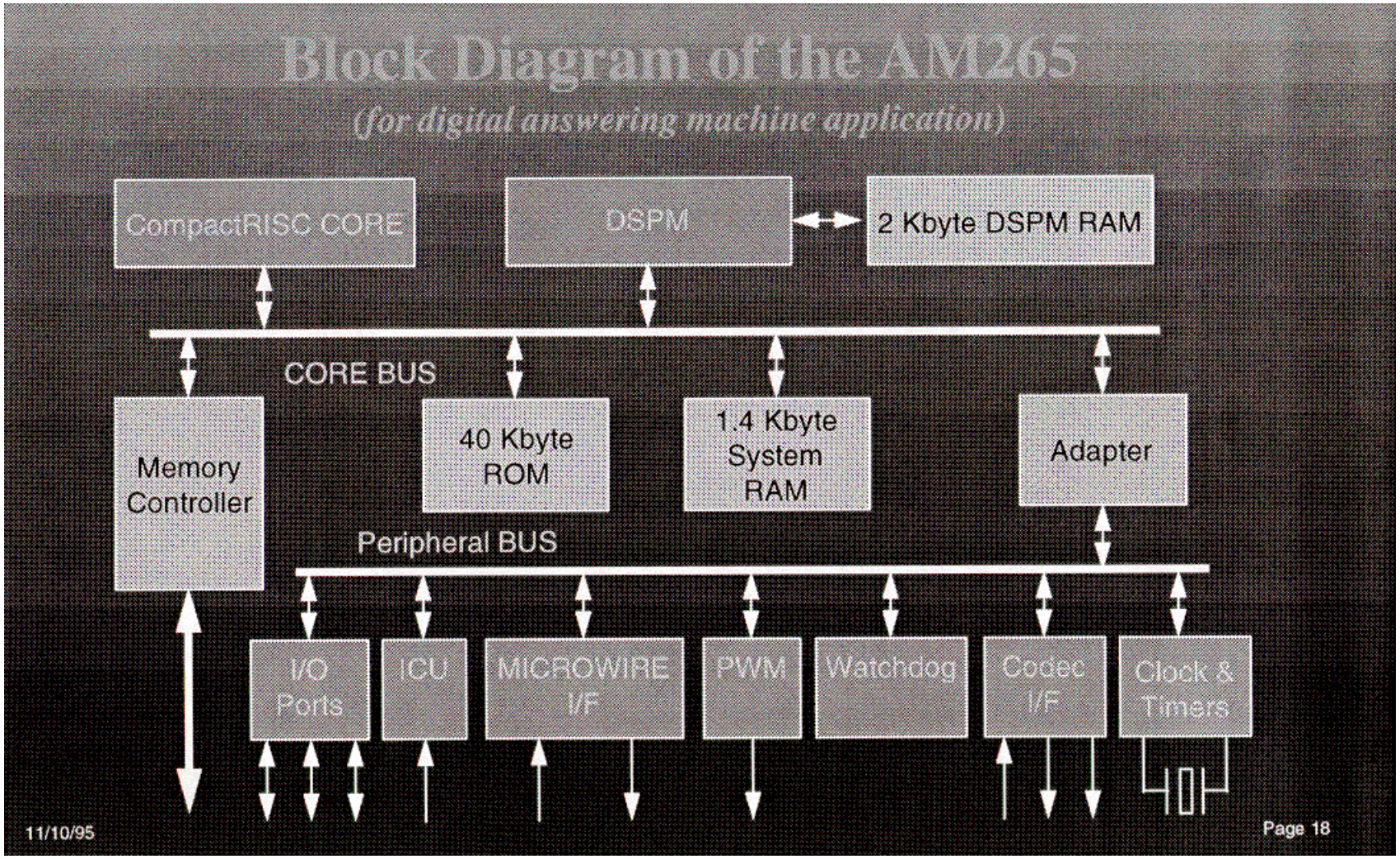
```
mov *r0,x0
mov *r1,y0
mpy x0, y0, a
add a, b
mov y0, *r2
inc r0
inc rl
```

Specialized Peripherals for DSPs

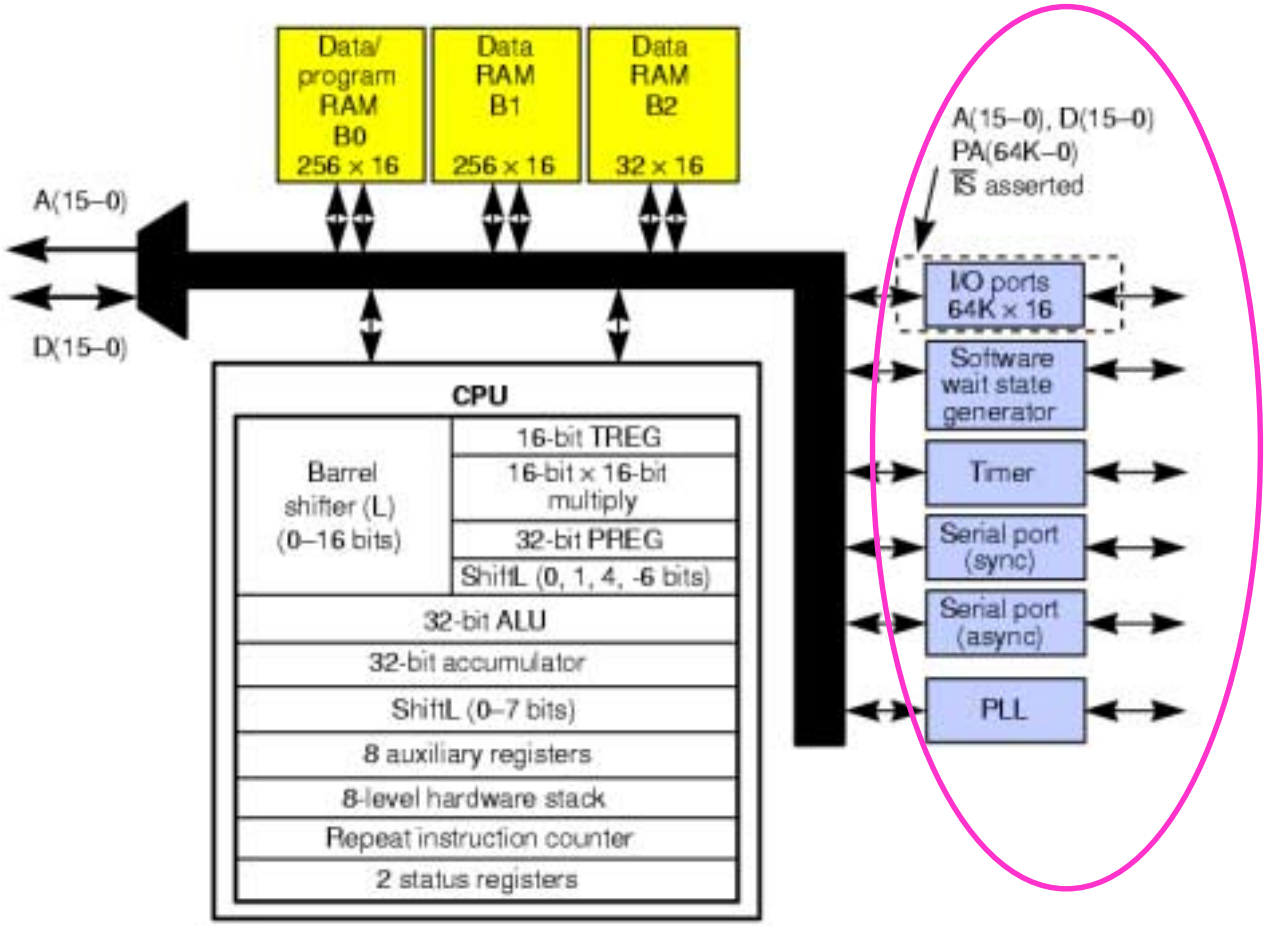
- **Synchronous serial ports**
- **Parallel ports**
- **Timers**
- **On-chip A/D, D/A converters**
- **Host ports**
- **Bit I/O ports**
- **On-chip DMA controller**
- **Clock generators**

- On-chip peripherals often designed for “background” operation, even when core is powered down.

Specialized peripherals



TMS320C203/LC203 BLOCK DIAGRAM DSP Core Approach - 1995



Summary of Architectural Features of DSPs

Data path configured for DSP

- **Fixed-point arithmetic**
- **MAC- Multiply-accumulate**

Multiple memory banks and buses -

- **Harvard Architecture**
- **Multiple data memories**

Specialized addressing modes

- **Bit-reversed addressing**
- **Circular buffers**

Specialized instruction set and execution control

- **Zero-overhead loops**
- **Support for MAC**

Specialized peripherals for DSP

THE ULTIMATE IN BENCHMARK DRIVEN ARCHITECTURE DESIGN!!!