Digital Signal Processors…

Number and variety of products that include DSP algorithms

+ Challenge of being cost effective, power/size effective → Special HWs

→ "So many dedicated ICs" → A little bit more general programmable DSP chips

Because of high cost of IC design, DSP processors became *less risky* and *famous* solutions, specially in low volume applications

DSP Algorithms Mold DSP Architectures

Every feature in a DSP processor is included to ease performing a DSP Algorithm

FIR filtering problem again → Each tap needs a multiply and add

1) Fast Multipliers

Processors perform multiplication by a series of shift/add operations

→ Needed multiple clocks cycles → To go for higher performance:

First commercially successful DSP TMS32010 included a single cycle Multiply or combined Multiply-Accumulate (MAC) unit, using a specialized HW

Almost all modern DSPs followed that

Digital Signal Processors…

2) Multiple Execution Units

Adding more independent execution units

for example a MAC unit + an ALU and a Shifter (Barrel Shifter: Single Cycle)

3) Efficient Memory Access → Higher Memory Bandwidth

Faster Memories + More Memories

a) Harvard Architecture instead of Von Neumann (see next slide figures)

Or modified Harvard…

Separate paths for data and program → Two memory access per clock cycle

single cycle access for single operand instructions

FIR needs more! → Two-operand instruction

b) Using Instruction Cache

Small block of RAM near the processor core

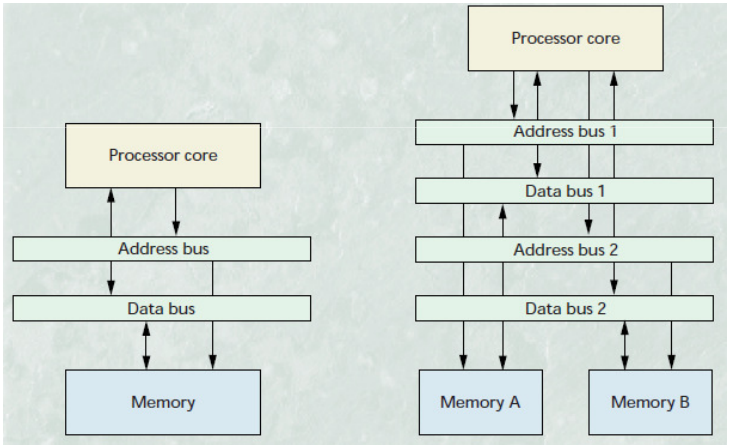
In loops (repeated instruction block) no need for instruction fetch → Another operand can be got from program memory

Digital Signal Processors…

Von Neumann vs. Harvard Architecture

Not just a matter of separate memories, but also separate data paths

→ Internal Memory …

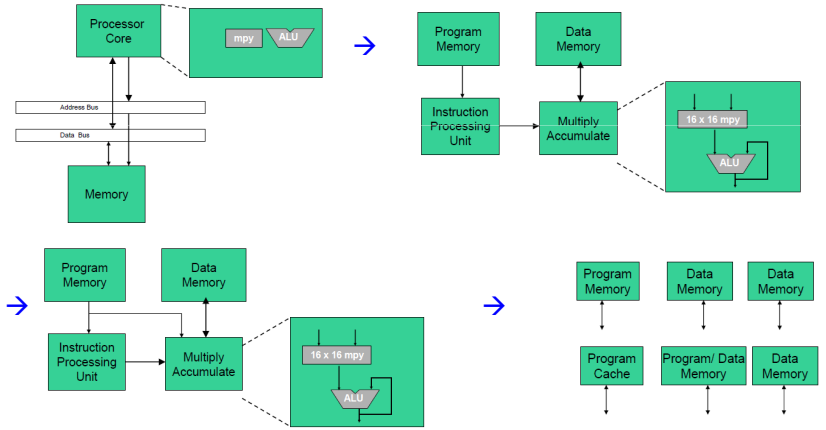


Digital Signal Processors…

Program Memory to Provide Filter Coefficients

Multiple Data Memories, Program Cache (at least one to have RPT command!)

Modified Harvard Architectures



Digital Signal Processors…

3) Efficient Memory Access…

Having predicted patterns of memory access in DSP algorithms leads to:

a) Dedicated HW for calculating memory addresses

b) Performs independently, parallel to other parts

c) Accessing new memory locations without pausing

Most common addressing mode in DSP processors

→ *Register indirect addressing with post increment*

→ *Having Circular addressing support in dedicated HW:*

Accessing a block of specified length sequentially and wrap around

As seen in FIR delay line

Circular Addressing is not part of C/C++ programming

→ *One of the reasons C compilers are not perfect for DSP processors*

Circular addressing is very helpful in implementing FIFO buffers for IO

Digital Signal Processors…

4) Data Format

Numeric fidelity needs careful attention in DSP algorithms

For example → Avoiding overflow, covering the dynamic range, …

→ *DSP applications are normally easier to implement in floating point format*

Sensitivity to cost and energy consumption force to:

→ *Fixed point DSPs*

→ *Smaller word width*

Conventional DSPs usually had 16-bit architecture, a few had 20, 24 and 32…

Instead, most conventional DSP processors:

a) Have wider Accumulator registers providing extra bits called 'guard bits' to extend the range of intermediate values

b) Support for saturation arithmetic, rounding and shifting to avoid overflow and maintain numeric fidelity

Digital Signal Processors…

5) Zero-Overhead Looping

Processing time of most DSP algorithms spent in loops

DSP architectures usually provide special support for loops

No extra clock cycles for updating the loop counter, testing the loop expiry and branching back

6) Streamed I/O

Processing time of most DSP algorithms spent in loops

DSP architectures usually:

a) provide special support for serial ports, parallel ports, memory interface, …

b) have special ports for specific streams, like TI McBSP to connect to audio codecs, or Video ports in new DSPs BT565x50 27.5 MHz

c) use DMA to allow data transfer with little or no intervention from computational units → Most ports in TI DSPs are equipped with DMA

Digital Signal Processors…

7) Specialized Instruction Set

DSP ISA design is a cost sensitive design

Conventional DSP ISAs are traditionally designed to:

a) Make maximum use of underlying HW → Maximum performance

Specifying multiple parallel operations in a single instruction, including several data fetches, multiple arithmetic operations, few address pointer updates, …

b) Minimize the program code size → Minimum program memory

Restricting usable registers in different operations, restriction of operations that can be done together, mode setting independent of the instruction (e.g. rounding and saturation modes)

→ *Highly specialized, complicated, and irregular instruction set*

Another reason for why DSP C/C++ compilers are not that efficient and low level programming and hand optimization are needed

Also C/C++ is not designed for describing DSP algorithms

Evolution of Digital Signal Processors

Conventional DSPs

based on conventional DSP architecture we have studied so far

1) Low cost, low performance range

Multiply or MAC + an ALU, address generator

Still in use → Very similar to DSPs of 80's

20-50 MHz clock

Examples: Motorola DSP560xx,

TI TMS320C2xx, Lucent DSP16xx

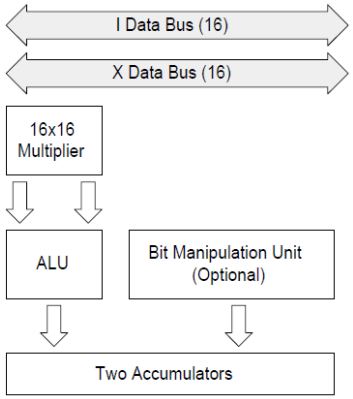
AD ADSP-21xx

Typical applications:

Consumer electronics,

Modest telecommunications products,

Hard disks



Evolution of Digital Signal Processors…

Conventional DSPs…

2) Midrange DSPs, higher performance

Thru higher clock rates and more sophisticated architectures, deeper pipeline

More HW units like barrel shifter, instruction cache, …

→ incremental (not dramatic) enhancement comparing to (1)

100-150 MHz clock

Examples:

Motorola DSP563xx,

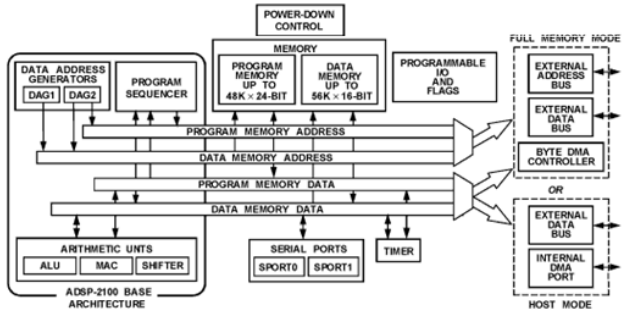
TI TMS320C54x,

AD ADSP-218x

Typical applications:

Consumer electronics,

Telecommunications products → VoIP, Wireless



Evolution of Digital Signal Processors…

Conventional DSPs… review

Issues with Old Conventional DSPs

- Slow External memory
- Pin count problem
- Expensive Internal memory
- Expensive HW and need for higher computational power

Design Goals :

1) To make maximum use of the processors underlying HW
2) To minimize the amount of memory space to store DSP programs

→ Short instructions but capable of a few memory fetch + some ALU ops instruction

→ Fewer number of registers coded in instructions

→ Using mode bits to control features instead of encoding them in instructions

→ Short but irregular, complicated and highly specialized instruction sets

→ Compilers are complicated , not friendly → Hand optimization is a must

Evolution of Digital Signal Processors…

Enhanced Conventional DSPs

Going beyond Faster Clock

Extending conventional DSP Architecture by adding more parallel execution units

Higher computational power, sometimes no high clock, no faster HW

Typically adding more ALUs or Multipliers or MAC units

e.g. Performing 2 MACs per cycle

- Extended instruction set to support new units
- Wider data buses to retrieve more data in parallel and feed new instruction

May need wider instruction words to include the parameters of new units in one instruction

Higher Cost and Complexity but Higher Performance

More advanced fabrication processes may help to justify the modifications

Evolution of Digital Signal Processors…

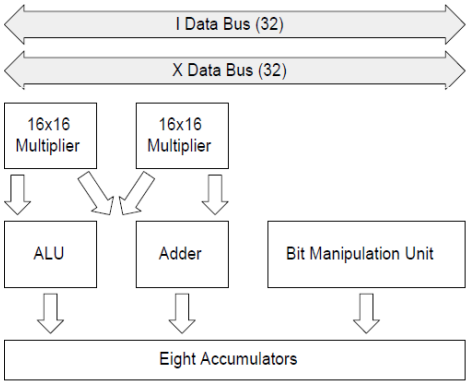
Enhanced Conventional DSPs…

Still Suffer from Conventional DSP Problems

- Complex Irregular instruction sets
- Hard to code and make compilers

Example:
Lucent DSP16XXX
Two MACs

Same range of applications



Evolution of Digital Signal Processors…

Multi Issue Approach

Design Goals :

1) Achieving Higher Performance
2) Instruction sets that lend themselves to compilers

→ *TI was the first to come up with a solution → TMS320C62XX , 1996*

→ *Then Motorola, Analog Devices, Lucent followed*

- *Very Simple Instructions, typically encoding a single operation per instruction*
- *Achieving high level of parallelism by issuing and executing instructions in parallel groups rather than one at a time*

→ Simplified instruction decoding and execution→ Higher clock rates

Many parallel execution units to execute instructions in parallel groups

→ High level of Parallelism and higher performance

→ Generally higher power consumption

Evolution of Digital Signal Processors…

Multi Issue Approach…

Multi Issue Processors → 1) Superscalar 2) VLIW ← Most *DSPs are VLIW*

Both have many execution units comparing to enhanced conventional DSPs

- **VLIW** DSPs issue a maximum of 4 to 8 instructions per cycle,

Fetched as a part of a long **super instruction**

→ *Grouping is done in programming time by compiler or programmer*

→ *Assembly language programmer or code generation tool specifies which instructions will be executed in parallel (at the time program is assembled)*

→ *Grouping will not change during execution*

→ *Wider instruction words, usually 32bit, to remove the restrictions on registers*

→ *Sufficient routers, buses and memory bandwidth are needed*

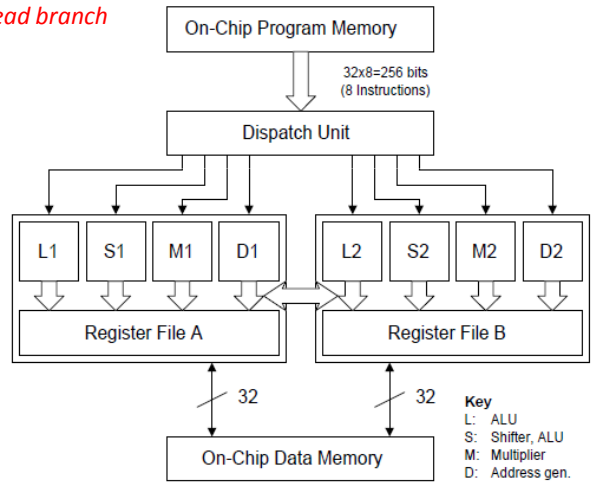
→ *Simpler instructions needs more instructions to do a task*

Evolution of Digital Signal Processors…

Multi Issue Approach…

Example: TI TMS320C62x Architecture…

No zero overhead branch



Evolution of Digital Signal Processors…

Multi Issue Approach…

- **Superscalar** processors typically issue less than VLIWs, say 2 to 4 instructions

→ *Differs from VLIW in how instructions are grouped*

→ *Grouping is done in run time and using specialized HW, and differs depending on the situation*

→ *Based on data dependencies and resource contention*

In superscalar processors…

→ *Burden of programming shifts from programmer to HW (easier to program)*

→ *Grouping of the same set of instructions may differ , for example in a loop*

→ *Difficult for programmers to predict how long given segments of code will take*

→ *For real time applications that is a disadvantage to meet time constraints*

→ *Extra cost /size/power for dynamic features*

Evolution of Digital Signal Processors…

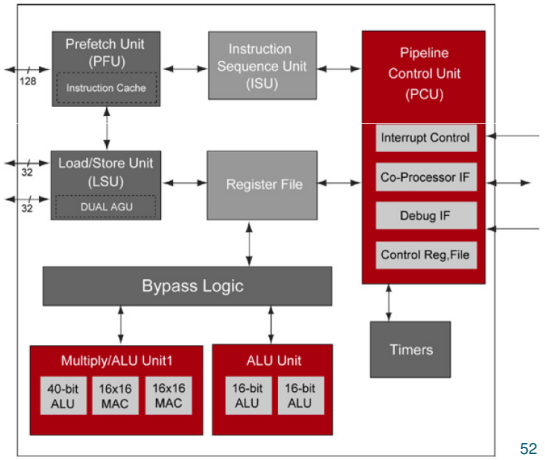
Multi Issue Approach…

Most of Multi-Issue DSPs are based on VLIW Architecture

As a rule DSPs traditionally avoid dynamic features

→ *Only a few Superscalar DSPs in the market*

e.g. ZSP500



Evolution of Digital Signal Processors…

SIMD

Single Instruction Multiple Data is not an Architecture

→ *An architectural technique that improves the performance*

Processor executes multiple instances of the same operation in parallel on different data

Two Methods:

1) More independent units with separate registers to work on different data at the same time

→ *e.g ADSP-2116x that has two basic ADSP2106x set of execution units including MAC, ALU, Shifter*

2) Logically Split their execution units into multiple sub units

For example an ALU to be used for 32bit add or two 16bit adds

→ *e.g ADS TigerSHARK is a VLIW DSP that has two sets of units and in each set split-ALU and Split-MAC can be used → Two levels of SIMD*

8 x16-bit multiplications can be done in single cycle

Evolution of Digital Signal Processors…

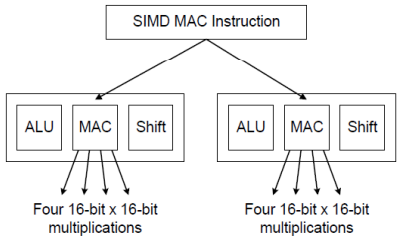
SIMD…

Difficult to program

→ *Needs programmer effort to modify the algorithms to fit into the architecture*

→ *Not possible for all algorithms*

→ *Specific applications*



Alternative to DSPs in some applications

- *High performance CPUs*
- *DSP/Micro-Controller Hybrids*

DSP Elements

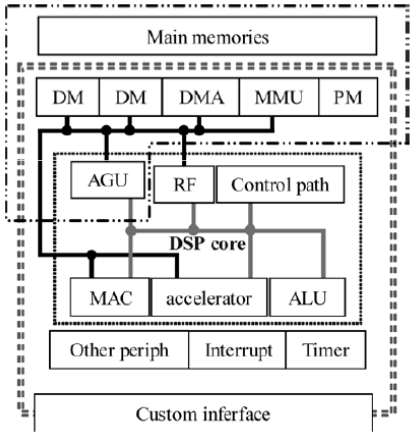
Inside a DSP

DSPs are based on RISC architecture with some CISC capabilities

1) Memory Subsystem
Internal Banks, Cache,
MMU, DMA...
External main memory

2) Processor Core
RF, GP ops, Special ops
Control path, Data path, AGU
using Data Bus (×n), Register Bus

3) Peripheral Subsystem
functional
SP, DMA, Interrupt, ...
non-functional
JTAG, ...



Digital Signal Processors History

- Intel 2920 Analog Signal Processor 1978 (no success)

Internal A2D and D2A + Simple signal Processor, no Hardware multiplier

- AMI S2811 Designed as a microcontroller peripheral 1979 (no success)

300 nsec MAC, Needed an MPU to initialize it

- NEC µPD7720, 1980

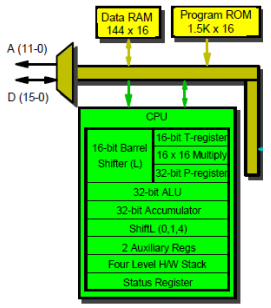
VLIW like instruction format, Internal RAM and ROM (128/512 words memory)

- AT&T DSP1 1980

MAC, internal memory, 20-bit fixed point data, 16 bit coefficient and program memory
800 nsec MAC

- TMS32010 Texas Instruments 1982

Using some of the ideas discussed...
160/200ns Instruction cycle time
390 nsec MAC
4K word external address reach
60 general purpose and
DSP specific instructions
Single cycle multiply
16-bit Barrel Shifter
External interrupt and polled input pins
Eight 16-bit I/O ports
40-pin DIP/44-pin PLCC



Digital Signal Processors History

- Hitachi CMOS DSP, HD61810 (HSP), 1982

First floating point DSP 12 bit Mantissa 4 bit Exponent, 250nsec MAC

- Fujitsu MB8764, 1983

Improved Fast Multiply/Accumulate (120 nsec)

- AT&T Bell Labs DSP32, 1984

The First 32 bit floating point DSP

- NEC µPD77230 , 1984

32 bit floating point DSP + 150nsec MAC

- TMS32020, TI, 1985

HW repeat command, 195nsec MAC

- ADSP-2100, 1986

16/40-bit Fixed point 125 nsec MAC, off-chip memory

- Fujitsu MB86232, 1987

First IEEE standard floating point DSP, 100 nsec MAC, Triple-Port Data Memory

- Motorola DSP56001, 1987

24/56-bit fixed point DSP, 74.1 nsec MAC

- TMS 320C25, TI, 1987

CMOS, Fixed-point 16/32, 100 nsec MAC

Digital Signal Processors History

- AT&T DSP16, 1987

Fixed-point 16/36, 55 nsec MAC

- AT&T DSP32C, 1988

CMOS, 32/40 floating point ,80 nsec MAC, Fast data memory, 2 access per cycle

- AT&T DSP16A, 1988

Fixed-point 16/36, 33 nsec MAC

- TMS320C30, TI 1988

CMOS, 32/40 bit Floating point, 60nsec MAC

- Analog Devices ADSP2101/2, 80 nsec MAC

Internal RAM and ROM

- Motorola DSP96002, 1989

32/64 IEEE floating point, 70 nsec MAC

.....

Old Era of DSPs ...

Applications:

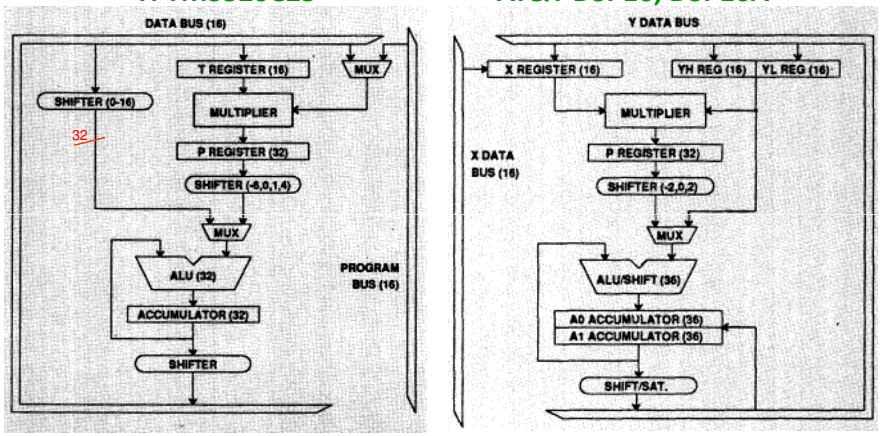
†**Voice-band:** 8KHz Sampling Rate → 125µsec sample period → 2500 × 50nsec MAC

† **Audio:** 44KHz Sampling Rate → 22.7µsec sample period → 454 × 50nsec MAC

Video: 5MHz Sampling Rate → 200nsec sample period → 4 × 50nsec MAC

Digital Signal Processors History

TI TMS320C25 | **AT&T DSP16, DSP16A**



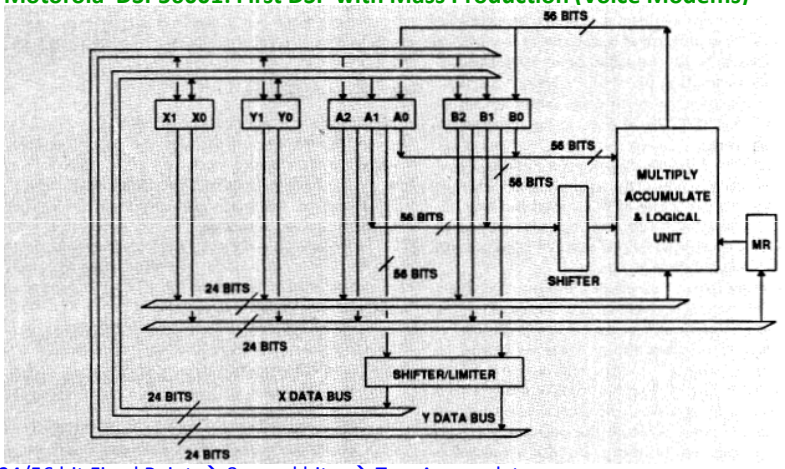
Fixed Point Arithmetic → Finite word Length→ Quantization Error → Scaling

More Precision → 36 bits Accumulators 2^4-1=15 → 32 bits can be added no overflow!

Shifters Saturation Arithmetic← When Overflow Occurs (OV flag)

Digital Signal Processors History

Motorola DSP56001: First DSP with Mass Production (Voice Modems)



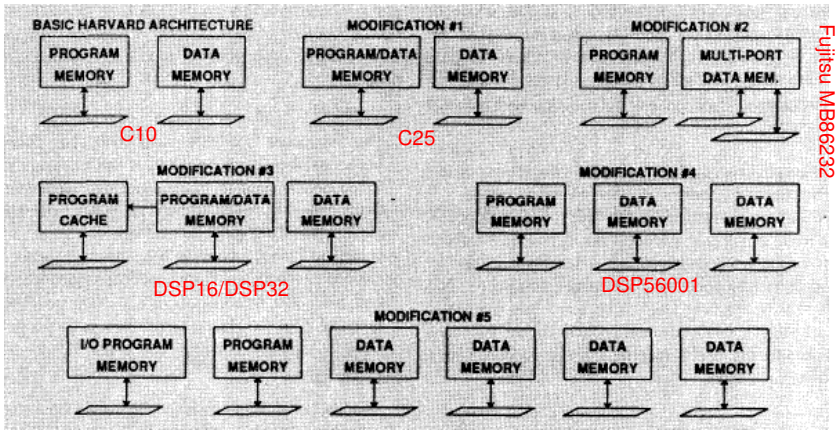
24/56 bit Fixed Point → 8 guard bits → Two Accumulators

No Product Register→ Multiply-Accumulate are done together

Mode Register : Part of Status Register = MR + CCR (Condition Code Register)

Digital Signal Processors History

DSP Memory Structures and Memory Band Width Concept

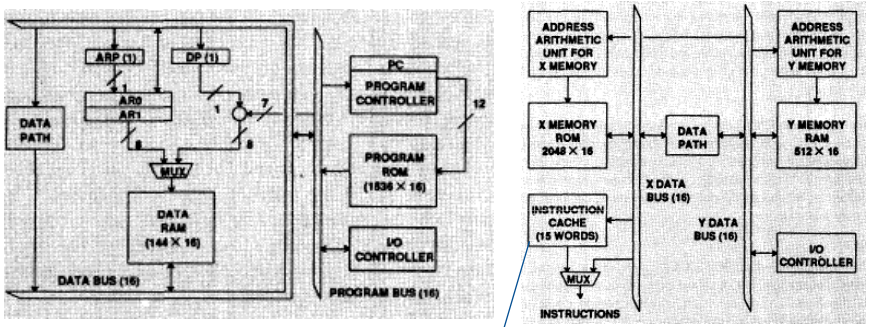


Cache for zero overhead looping

Conflicts might happen

Digital Signal Processors History

TMS320C10, Basic Harvard | **DSP16(A), Modification#3**



memory		cache		
0	a0	0	a2	2
1	a1	1	a0	0
2	a2			tag

Cache tag

repeat command

if miss tag→ add entry

Needs a Replacement Policy → e.g. LRU (Least Recently Used)

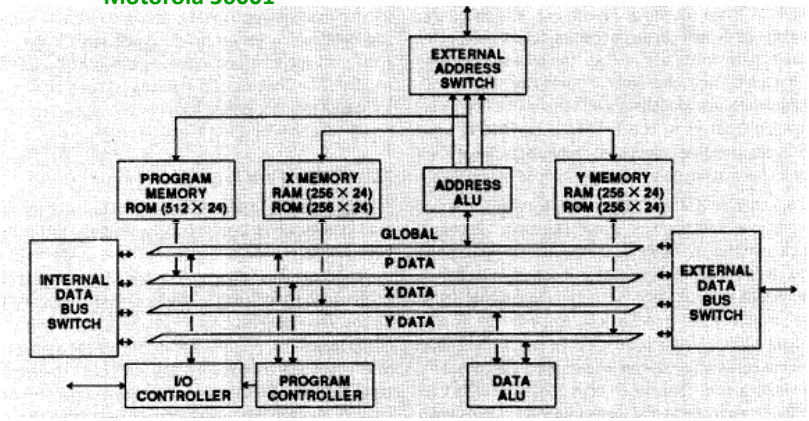
if hit tag → Reads or Writes from/to Cache

Write policy is needed → e.g. Write-Thru Cache (every write to cache causes a write to memory)

Policies must be simple to be implemented in small/quick circuits

Digital Signal Processors History

Motorola 56001



3 paths X-Y data and P program → Sine-wave table in Y-ROM, A/μ law in X-ROM

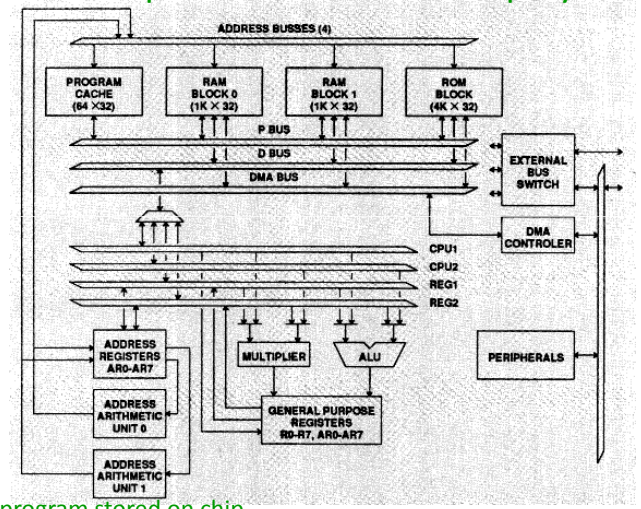
Modification 4 → no cache but zero overhead looping

Cache is extended to become a memory bank

Simultaneous fetch of an instruction and 2 operands

Digital Signal Processors History

TMS320C30: parallel memories and 2 accesses per cycle memories



Only if program stored on chip

Digital Signal Processors Design Ideas

Summary

- Hardware modulo addressing (Circular addressing), bit reversed addressing, ..
- DMA + Fast Internal Memory
- Multiple arithmetic units + memory architectures to support several accesses per instruction cycle (specialized instruction set)
- Separate program and data memories, Harvard Architecture
- Special SIMD (single instruction, multiple data) operations
- VLIW techniques so each instruction drives multiple arithmetic units in parallel
- Special arithmetic operations, (e.g. MAC)
- Zero overhead looping, Special HW loop controls, without overhead for instruction fetches or exit testing
- No memory management (virtual memory, memory protection, memory switching, ...) and other GPP OS memory stuff
- Wide registers with special precision functionalities, saturation, guard bits, ...
- Data/Program Cache Memory