DSP Architecture
Introduction

- Multi-processor vs. Multi-computer architecture
- μP vs. DSP
- RISC vs. DSP

**RISC**
- Reduced-instruction-set
- Register-to-register operation
- Higher throughput by using highly pipelined execution of simple instructions and efficient compilers.

**CISC**
- Complex-instruction-set
Standard DSP architecture

- A fast on-chip multiplier
- An instruction cycle is generally one or two clock cycles long.
- Several functional units that perform several parallel operations. The functional units typically include ALU, two or more address generator, etc.
- Several on-chip memory units used to store instructions, data, or look-up table.
- Large numbers of temporary registers are used to store values used for long periods of time.
- Several on-chip system buses to increase memory bandwidth.
- Support for special addressing modes, especially modulo and bit-reversed addressing.
Introduction to DSP

Classification

- Von Neumann architecture
- Harvard architecture

TMS32010

- Each MAC requires at least two cycles since the two multiplier operands must be fetched sequentially
- The performance is limited by the single bus
- 5 MOPs, i.e., 200ns instruction cycle time

TMS320C25 & C50

- Single-cycle MAC
- Harvard architecture is used
- C50 is an enhanced version of C25 with twice throughput
Introduction to DSP (cont’d)

TMS320C30
- Multi-bus 320-bit floating-point DSP
- It runs at 50MHz and can perform up to eight operations per instruction cycle, which corresponds to 200 MFLOPS (clock is divided by 2 internally)

Motorola DSP 56001 and 56002
- Triple-bus Harvard architecture
- Two clock cycles are required per instruction
- It runs at 40MHz, which corresponds to 50ns instruction cycle time.
- 56002 is an enhanced, software compatible, version of 56001
- 56002 can perform up to six operations simultaneously, which corresponds to 120MOPS@40MHz (VLIW)
- VLIW architecture
  - A parallel architecture that uses multiple, independent functional units and packages multiple operations into one very long instruction.
Introduction to DSP (cont’d)

- Motorola 96001 and 96002
  - A floating-point version of the 56001
- Ideal DSP architecture
  - Processing elements
  - Storage elements
  - Interconnection networks
  - Control
Introduction to DSP (cont’d)

- **Multi-processors**
  - Tightly coupled system: all processors share the same common main memory
  - Loosely coupled system: the main memory is distributed among the processors, although the processors share the same memory space

- **Multi-computers**
Introduction to DSP (cont’d)

Message-based architectures

- SISD: classical serial computer
- MISD: impractical
- SIMD: suitable for applications with high parallelism, such as image processing
- MIMD: complex design

Interconnection topologies

- Crossbar: complete interconnection between all processor induce high cost
- Single-bus: inefficient
- Packet switching vs. circuit switching
Introduction to DSP (cont’d)

- Shared-memory architecture
  - Memory bandwidth bottleneck
  - Imbalance between computation capacity and communication bandwidth
  - \( T_{pe} \geq 2NT_m \)

- Reducing the memory cycle time
  - Interleaving memory
  - Cycle time is reduced by a factor of \( K \). If choosing \( K=2N \), a good balance is obtained.

- Reducing communication
  - Broadcasting
    - All PE operate on the same input data
    - Hence, only one memory read cycle is needed
  - Interprocessor communication
  - Cache memory
Figure 8.24  Dataparallel multiprocessor architecture

Figure 8.25  Processor architecture for the PEs
Figure 8.26 Multiprocessor architecture

Figure 8.27 Interleaving of K memories

Figure 8.28 Broadcasting of data

Figure 8.29 Architecture with direct interprocessor communication

Figure 8.30 FFT architecture with a single butterfly
### Table 1-RISC VS DSP floating-point CPUs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Intel i860 RISC CPU</th>
<th>Motorola DSP96002 and Texas Instruments TMS320C40 32-bit DSP CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed to run Unix</td>
<td>Designed to run Unix</td>
<td>Won't run Unix, no MMU</td>
</tr>
<tr>
<td>64-bit arithmetic</td>
<td>64-bit arithmetic</td>
<td>32-bit arithmetic</td>
</tr>
<tr>
<td>25-nsec instruction cycle (pipelined)</td>
<td>25-nsec instruction cycle (pipelined)</td>
<td>50-nsec instruction cycle (pipelined)</td>
</tr>
<tr>
<td>No DMA for concurrent I/O</td>
<td>No DMA for concurrent I/O</td>
<td>DMA controllers for concurrent I/O</td>
</tr>
<tr>
<td>Slow context switch</td>
<td>Slow context switch</td>
<td>Faster context switch</td>
</tr>
<tr>
<td>Standard interrupt mechanism</td>
<td>Standard interrupt mechanism</td>
<td>Faster interrupts</td>
</tr>
<tr>
<td>One external bus</td>
<td>One external bus</td>
<td>Two external buses</td>
</tr>
<tr>
<td>Off-chip memory penalty</td>
<td>Off-chip memory penalty</td>
<td>Single-cycle external memory</td>
</tr>
<tr>
<td>No cache lock</td>
<td>No cache lock</td>
<td>Single-cycle external memory</td>
</tr>
<tr>
<td>Pipelined FPU</td>
<td>Pipelined FPU</td>
<td>Single-cycle FPU multiply operations</td>
</tr>
<tr>
<td>No integer-to-floating-point conversion instruction</td>
<td>No integer-to-floating-point conversion instruction</td>
<td>Single-cycle integer-to-floating-point conversion</td>
</tr>
<tr>
<td>No integer multiples</td>
<td>No integer multiples</td>
<td>32 * 32-bit integer MAC</td>
</tr>
<tr>
<td>Graphics engine</td>
<td>Graphics engine</td>
<td>No graphics engine</td>
</tr>
<tr>
<td>Need high-level language</td>
<td>Need high-level language</td>
<td>Easy-to-use assemblers</td>
</tr>
<tr>
<td>C compilers</td>
<td>C compilers</td>
<td>C compilers optimized for parallel operations</td>
</tr>
<tr>
<td>Long-instruction-word operation; as many as 2 instructions/word, 1 integer, 1 floating point</td>
<td>Long-instruction-word operation; as many as 2 instructions/word, 1 integer, 1 floating point</td>
<td>1 instruction/word parallel operation</td>
</tr>
</tbody>
</table>

**S/N contribution**

- 1 bit $\approx 6\text{dB}$
- 12 bits $\approx 72\text{dB}$
- 16 bits $\approx 96\text{dB}$
Modified Harvard Architecture

program/data memory

data memory

two simultaneous fetches
Modified Havard Architecture (cont’d)

- Instruction cache
- Program/data memory
- Data memory

Simultaneous fetches when cache is used
Example
Modified Havard Architecture (cont’d)

 simultaneuously fetch instruction and two operands
Example (DSP56001)
Example (TMS320C30)
## Memory Summary

<table>
<thead>
<tr>
<th>part</th>
<th># of banks</th>
<th>demand ratio</th>
<th>instr. cache</th>
<th>internal memory</th>
<th>external space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP32</td>
<td>2</td>
<td>4</td>
<td>no</td>
<td>(2)512W data RAM 512W prog ROM</td>
<td>14KW data/prog</td>
</tr>
<tr>
<td>DSP32C</td>
<td>2</td>
<td>4</td>
<td>no</td>
<td>(2)512W data RAM 512W prog ROM or 1KW prog ROM</td>
<td>4MW data/prog</td>
</tr>
<tr>
<td>DSP16</td>
<td>2</td>
<td>2</td>
<td>15W</td>
<td>512W data RAM 2KW prog/data ROM</td>
<td>64KW program</td>
</tr>
<tr>
<td>DSP16A</td>
<td>2</td>
<td>2</td>
<td>15W</td>
<td>2KW data RAM 4KW prog/data ROM</td>
<td>64KW program</td>
</tr>
<tr>
<td>DSP56001</td>
<td>3</td>
<td>3</td>
<td>no</td>
<td>(2)256W data RAM (2)256W data ROM 512W prog RAM</td>
<td>(3)64KW prog/data</td>
</tr>
<tr>
<td>DSP96001</td>
<td>3</td>
<td>4</td>
<td>no</td>
<td>512W prog RAM (2)512W data RAM (2)512W data ROM</td>
<td>4GW prog (2)4GW data</td>
</tr>
<tr>
<td>TMS32010</td>
<td>2</td>
<td>2</td>
<td>no</td>
<td>144W data RAM 1.5KW prog ROM</td>
<td>4KW prog</td>
</tr>
<tr>
<td>TMS32020</td>
<td>2</td>
<td>2</td>
<td>1W</td>
<td>288W data RAM 256W prog/data RAM</td>
<td>64KW data</td>
</tr>
<tr>
<td>TMS320C25</td>
<td>2</td>
<td>2</td>
<td>1W</td>
<td>288W data RAM 256W prog/data RAM 4KW prog ROM</td>
<td>64KW data</td>
</tr>
<tr>
<td>TMS320C30</td>
<td>3</td>
<td>4</td>
<td>64W</td>
<td>(2)1KW prog/data RAM 4KW prog/data ROM</td>
<td>16MW data/ prog</td>
</tr>
</tbody>
</table>
Basic Addressing Modes

- **Immediate**: The operand is part of the instruction
- **Direct**: A memory address is part of the instruction
- **Indirect**: An auxiliary register contains the memory address
More Elaborate Addressing Modes

- **Modulo mode**: Used to implement circular buffers, queues, or delay lines, such as those in FIR filters
- **Indexed**: An index is added to an address *before* the memory access
- **Bit reversed**: Used to compute FFTs
Pipelining

- Is it there? Is it visible?
- Reservation tables
- Time stationary vs. Data stationary code vs. interlocking
A Common Pipeline Model

- 3 distinct programming models for pipelining:
  - Interlocking
  - Time stationary coding
  - Data stationary coding
A More Elaborate Pipeline Model
Branching

- There may not be sufficient time between instruction fetches to decode a branch instruction before the next instruction is fetched.
- If the program address space is large, the destination address may not fit in an instruction word, so a second fetch from the instruction memory may be required. Alternatives are paging or PC-relative addressing.
- In the case of conditional branching, the fetch of the next instruction cannot occur before the condition codes in the ALU can be tested.
Solutions

- Multi-cycle branch instructions
- Delayed branches
- Low overhead looping
One Way to Deal with Pipelining

- Use a disjoint set of registers for each task
- The control structure of the tasks is tied together however
Pipeline Interleaving
Pipeline Interleaving (cont’d)

- Programmer (or compiler) sees multiple parallel processors that share memory without contention.
- The assembly language is unaffected by the amount of pipelining.
- It would help to have software to automate task partitioning and synchronization.
Towards Macro Parallelism

- Support mechanisms currently in place
  - DMA (use excess memory cycles)
  - Hardware controlled wait states (allows for transparent contention resolution)
  - Bit test-and-set operations (allows shared data structures)
  - Dual external busses (allows private and shared memory)
  - Chip-level simulators (allows system simulation)
Real DSP Hardware

- Cover the architecture design in block level
- Elements
  - Multiplier / Accumulator
  - ALU
  - Shifter
  - DAG
  - Program Sequencer
  - Memory
  - Communication (DMA)
System Selection & Design

- Hardware system consideration

Selection Procedure

- Selecting Hardware
- High-level symbols design
- Component organization
- Bit-level design
Data Address Generator

General scheme of data-address generator
The ADSP-1410 Data-Address Generator
Program Sequencer

- Instruction sequencer and its relationship to other DSP system component
Program Sequencer (cont’d)

Simple sequencer control structure

- A then B
- If W then A else B
- While W do A
Functional organization of a powerful program sequencer (ADSP-2100)

- Next (program) - address Multiplexer

![Diagram showing the functional organization of a program sequencer](image)
Program Sequencer (cont’d)

- Next address selection
- Condition logic

Diagram showing the flow of data and control signals in a program sequencer, including components like the status register, instruction register, condition logic, and control signals for address selection and loop management.
Program Sequencer (cont’d)

A word-slice micro-programmable program sequencer (ADSP-1401)
System Selection

Micro-coded system

Overview
System Selection (cont’d)

Micro-coded system (cont’d)

- Micro-code word broken up into sections, each an instruction to an individual component
System Selection (cont’d)

- Fast FIR filter structure employing micro-programmed parallel computing architecture
System Selection (cont’d)

- Design Example: FIR filter chip
  - Hardware implementation of an FIR chip
System Selection (cont’d)

Design Example: FIR filter chip (cont’d)

FIR filter memory organization for coefficients and data

![Diagram of FIR filter memory organization for coefficients and data]
System Selection (cont’d)

- Am29500 array processor, a high-performance micro-programmed complex signal processor
System Selection (cont’d)

Throughput implementation by custom paralleling

- \( AR' = AR + BR \times WR - BI \times WI \)
- \( AI' = AI + BI \times WR + BR \times WI \)
- \( BR' = AR - BR \times WR + BI \times WI \)
- \( BI' = AI - BI \times WR - BR \times WI \)

The system requires

- Input of A and B, two complex or 4 real number inputs
- Four multiplications: \( BR \times WR, BI \times WI, BR \times WI, BI \times WR \)
- Four summations of the form \((a \pm b \pm c)\), or six simple summations
- Scaling down by two to prevent overflow
- Output of A’ and B’, two complex (4 real) outputs
System Selection (cont’d)

- Effect on throughput and resource utilization of paralleled components in a FFT spectrum analyzer
  - Allocation

<table>
<thead>
<tr>
<th>Resource →</th>
<th>Data-memory</th>
<th>ALU</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations needed per butterfly</td>
<td>8</td>
<td>8 (6?)</td>
<td>4</td>
</tr>
<tr>
<td>Options:</td>
<td>Buses</td>
<td>Cycles in use</td>
<td>ALUs</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
System Selection (cont’d)

♦ Scheduling

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Data Bus = ALU-IN</th>
<th>Multiplier Inputs</th>
<th>Multiplier Output</th>
<th>Real ALU Operation</th>
<th>Imag ALU Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read B</td>
<td>BR, WR</td>
<td>BR*WR</td>
<td>AR – BR*WR</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Read A</td>
<td>BR, WI</td>
<td>BR*WI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>BR*WR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>BR*WI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Write B’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Write A’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DSP Microprocessor - Architecture

Block diagram of TMS32020
Arithmetic-logic-unit block diagram
Multiplier-accumulator section of ADSP-2100
DSP Microprocessor – Architecture (cont’d)

Block diagram of ADSP-2100’s shifter
DSP Microprocessor – Architecture (cont’d)

Data-Address-Generator block diagram
DSP Microprocessor – Architecture (cont’d)

Block diagram of ADSP-2100’s program sequencer