11.4 Von Neumann Architecture

In 1946, John Von Neumann developed the first computer architecture that allowed the computer to be programmed by codes residing in memory. In this, program instructions were stored in Read Only Memory (ROM). The Von Neumann architecture is most widely used in majority of microprocessors. In a computer with Von Neumann architecture, the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instruction and data use the same signal pathways and memory. The Von Neumann architecture consists of three buses. The data bus, the address bus and control bus.

The Data bus: Transports data between CPU and its peripherals. It is bidirectional. The CPU can read or write data in the peripherals.

The address bus: The CPU uses the address bus to indicate which peripherals it wants to access and within each peripheral which specific register. The address bus is unidirectional. The CPU always writes the address, which is read by the peripherals.

Control bus: The bus carrier signals that are used to manage and synchronize the exchanges between the CPU and its peripherals, as well as that indicates if the CPU wants to read or write the peripheral.

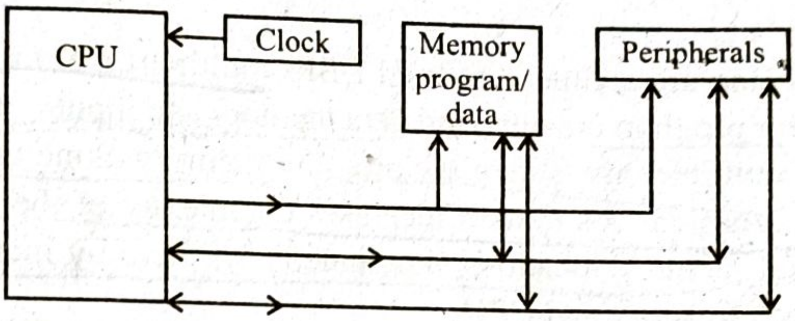

Fig. 11.1 Von Neumann Architecture

The main characteristics of the Von Neumann architecture is that it only possesses 1 bus system. The same bus carries all the information exchanged between the CPU and the peripherals including the instruction codes as well as the data processed by the CPU.

11.5 Harvard Architecture ✓

The term Harvard originated from the Harvard Mark 1 relay-based computer which stored instructions on punched tape and data in relay latches. The Harvard architecture physically separates memories for their instructions and data, requiring dedicated buses for each of them. Instructions and operands can therefore be fetched simultaneously.

Most DSP processors use a modified Harvard architecture with two or three memory buses; allowing access to filter coefficients and input signals in the same cycle. Since it possesses two independent bus systems, the Harvard architecture is capable of simultaneous reading an instruction code and reading or writing a memory or peripheral as part of the execution of the previous instruction. Since it has two memories, it is not possible for the CPU to mistakenly write codes into the program memory and therefore compute the code while it is executing.

However it is less flexible. It needs two independent memory banks. These two resources are not interchangeable.

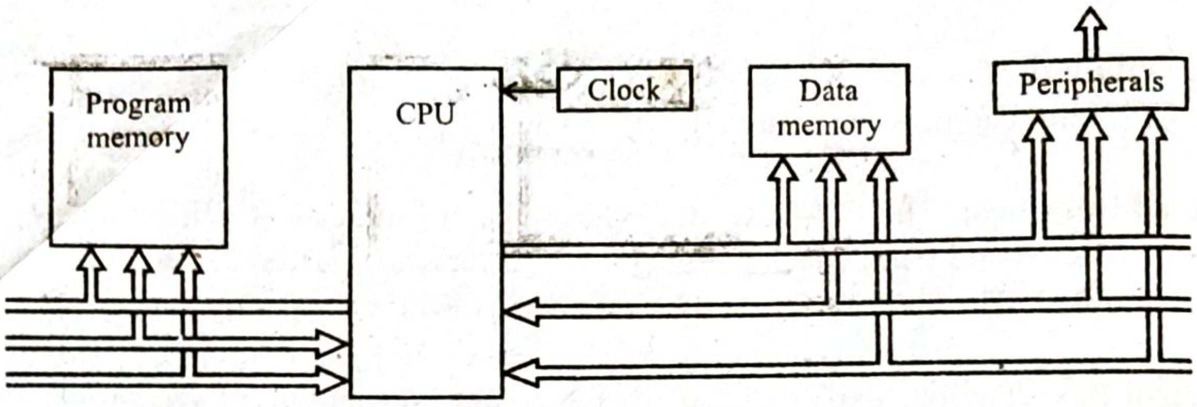

Fig. 11.2 Harvard architecture

The modified Harvard architecture used DSPs multiport memory that has separate bus systems for program memory and data memory and input/output peripherals It may also have multiple bus system for program memory alone or for data memory alone. These multiple bus system increases complexity of the CPU, but allow it to access several memory locations simultaneously, there by increasing the data throughput between memory and CPU.

11.6 VLIW Architecture

The new architecture that has attracted a great deal of attention in the DSP community is the Very Long Instruction Word(VLIW). The Very long instruction word processing increase the number of instructions that are processed per cycle. It is essentially a concatenation of several short instructions and require multiple execution units, running in parallel, to carry out the instructions in a single cycle. The new architecture makes use of extensive parallelism whilst retaining some of the good features of previous DSP processors. VLIW architecture executes multiple instructions/cycle and use simple, regular instruction sets.

The Very Long Instruction Word (VLIW) processor consists of architecture that reads a relatively large group of instructions and executes them at the same time. The VLIW processor combines many simple instructions into a single long instruction word that uses different registers. A language compiler or pre-processor separates program instructions into basic operations that are performed by the processor in parallel . These operations are placed into a "very long instruction word" that the processor can then disassemble, and then transfer each operation to an appropriate execution unit. For example, the group might contain four instructions, and the compiler ensures that those four instructions are not dependent on each other so they can be executed simultaneously. Otherwise, it places "no-ops" (blank instructions) in the group where necessary.

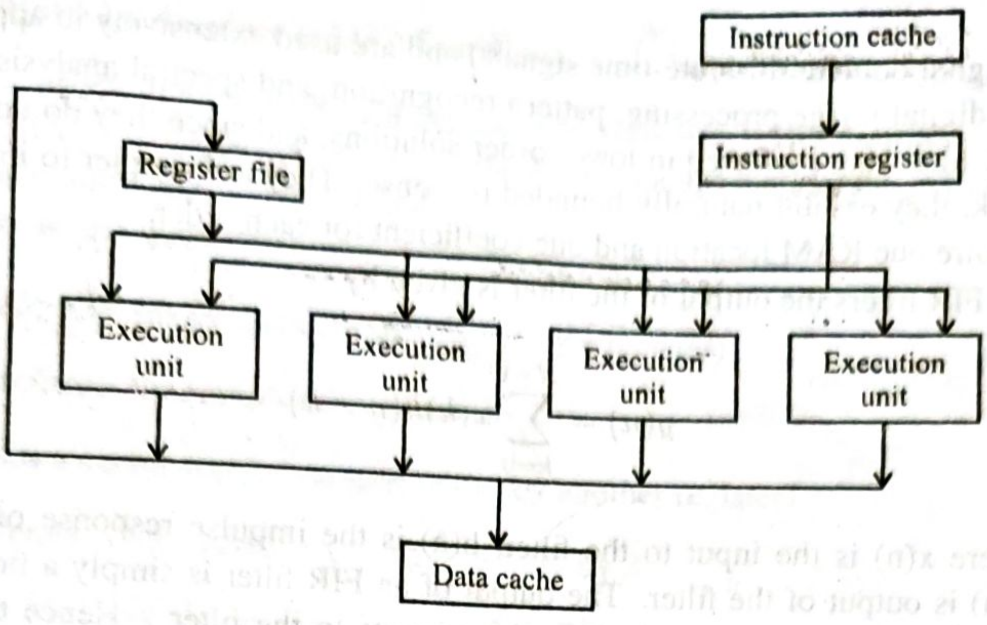

Fig. 11.3 VLIW architecture

Advantages of VLIW architecture

1. Increased performance
2. Better compiler targets
3. Potentially easier to program
4. Potentially scalable
5. Can add more execution units, allow more instructions to be packed into the VLIW instruction.

Disadvantages of VLIW architecture

1. New kind of programmer/ compiler complexity
2. Program must keep track of instruction scheduling
3. Increased memory use
4. High power consumption
5. Misleading MIPS ratings

11.7 Multiply Accumulate Unit (MAC)

The Multiply-Accumulate (MAC) operation is the basis of many digital signal processing algorithms, notably digital filtering. The term "digital filter" refers to an algorithm by which a digital signal or sequence of numbers is transformed into a another sequence of numbers termed the output digital signal. Digital filters involve signals in the digital domain (discrete-time signals) and are used extensively in applications such as digital image processing, pattern recognition, and spectral analysis. In general FIR filters are preferred in lower order solutions, and since they do not employ feedback, they exhibit naturally bounded response. They are simpler to implement, and require one RAM location and one coefficient for each order.

For FIR filters the output of the filter is given by

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k) \tag{11.1}$$

where x(n) is the input to the filter, h(n) is the impulse response of the filter and y(n) is output of the filter. The output of an FIR filter is simply a finite length weighted sum of the present and previous inputs to the filter . Hence to perform filtering through above equation, the minimum requirement is to quickly multiply two values, and add the result. To make it possible, a fast dedicated hardware MAC, using either fixed point or floating point arithmetic is mandatory. Characteristics of a typical fixed point MAC include

1. 16 × 16 bit 2's complement inputs
2. 16 × 16 bit multiplier with 32-bit product in 25 ns
3. 32/40 bit accumulator

In the TMS320C50, for example, the FIR equation can be efficiently implemented using the instruction pair:

```
RPT     NM1
MACD    HNM1, XNM1
```

The first instruction, RPT NM1, loads the (N-1) into the repeat instruction counter, and causes the multiply-accumulate with data move (MACD) instruction following it to be repeated N times. The MACD instruction performs a number of operations in one cycle:

1. Multiplies the data sample, $x(n-k)$, in the data memory by the coefficient, $h(k)$, in the program memory;
2. Adds previous product to the accumulator;
3. Implements the unit delay, symbolized by z^{-1}, by shifting the data sample, $x(n-k)$, up to update the tapped delay line.

The Multiply-Accumulate (MAC) Function

The MAC speed applies both to finite impulse response (FIR) and finite impulse response (IIR) filters. The complexity of the filter response dictates the number MAC operations required per sample period.

A multiply-accumulate step performs the following:

- Reads a 16-bit sample data (pointed to by a register)
- Increments the sample data pointer by 2
- Reads a 16-bit coefficient (pointed to by another register)
- Increments the coefficient register pointer by 2
- Sign Multiply (16-bit) data and coefficient to yield a 32-bit result
- Adds the result to the contents of a 32-bit register pair for accumulate.

The TMS320C54x multiply-accumulate (MAC) unit performs a $16 \times 16 \longrightarrow 32$-bit fractional multiply-accumulate operation in a single instruction cycle. The multiplier supports signed/signed multiplication, signed/unsigned multiplication, and unsigned/unsigned multiplication. These operations allow efficient extended-precision arithmetic. Many instructions using the MAC unit can optionally specify automatic round-to-nearest rounding.

11.8 Pipelining

Most of the early microprocessors execute instructions entirely sequentially. After the execution of first instruction the next one starts. The problem with this is that it is extremely inefficient, since the second instruction has to wait until all the steps of first instruction are completed. To improve the efficiency, advanced microprocessors and digital signal processors use an approach called pipelining in which different phases of operation and execution of instructions are carried out in parallel. That is in modern processors the first step of execution is performed on the first instruction, and then when the instruction passes to the next step, a new instruction is started. The steps in the pipeline are often called stages.

The basic action of any microprocessor can be broken down into a series of four simple steps. They are

1. **The Fetch** phase(F) in which the next instruction is fetched from the address stored in the program counter.
2. **The decode** phase (D) in which the instruction in the instruction register is decoded and the address in the program counter is incremented.

3. **Memory read** (R) phase reads the data from the data buses and also writes data to the data buses.

4. **The Execute phase (X) executes** the instruction currently in the instruction register and also completes the write process.

In a modern processor, the above four steps get repeated over and over again until the program is finished executing. These are, in fact, the four stages in a classic RISC pipeline. Each of the above stages could be said to represent one phase in the "lifecycle" of an instruction. An instruction starts out in the fetch phase, moves to the decode phase, then to the memory read phase, and finally to the execute phase. Each phase takes a fixed, but by no means, equal amount of time.

Pipelining a processor means breaking down its instruction into a series of discrete pipeline stages which can be completed in sequence by specialized hardware. Because an instruction's lifecycle consists of four fairly distinct phases, the instruction execution process is divided into a sequence of four discrete pipeline stages, where each pipeline stage corresponds to a phase in the standard instruction lifecycle. Note that the number of pipeline stages is referred to as the pipeline depth. So a four-stage pipeline has a pipeline depth of four.

To understand the pipelining in a better way, let us assume that the number of stages is four and the execution time of an instruction is four nanoseconds. If we assume the time taken for each stage in the instruction is equal, then the time taken for each stage is one nanosecond. So our original single-cycle processor's four-nanosecond execution process is now broken down into four discrete, sequential pipeline stages of one nanosecond each in length. At the beginning of the first nanosecond, the first instruction enters the fetch stage. After that nanosecond is complete, the second nanosecond begins and the first instruction moves on to the decode stage while the second instruction enters the fetch stage. At the start of the third nanosecond, the first instruction advances to the memory read stage, the second instruction advances to the decode stage, and the third green instruction enters the fetch stage. At the fourth nanosecond, the first instruction advances to the execution stage, the second to the memory read stage, the third to the decode stage, and the fourth to the fetch stage. After the fourth nanosecond has fully elapsed and the fifth nanosecond starts, the first instruction has passed from the pipeline and is now finished executing. Thus we can say that at the end of four nanoseconds (= four clock cycles) the pipelined processor depicted below has completed one instruction. At start of the fifth nanosecond, the pipeline is now full and the processor can begin completing instructions at a rate of one instruction per nanosecond. This 1 instruction/ns completion rate is a four-fold improvement over the single-cycle processor's completion rate of 0.25 instructions/ns (or 4 instruction every 16 nanoseconds).

The pipelining stages for different DSPs are shown in table 11.2. Note that TMS320C54x has two additional phases : pre-fetch (PF) phase which stores the address of the instruction to be fetched and the access phase (A) which reads the address of the operand and modify the auxiliary registers and stack pointer if required.

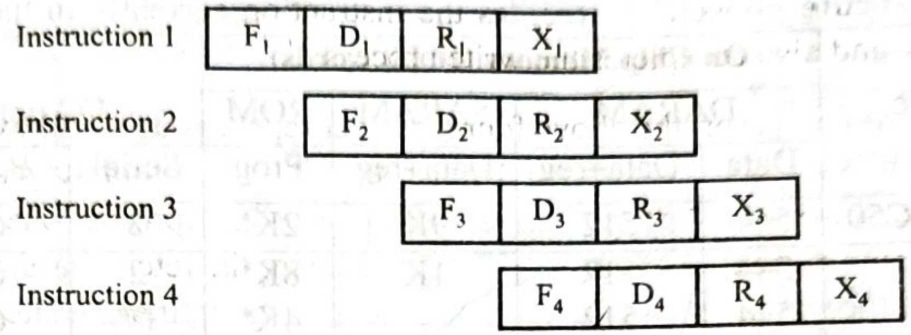

Fig. 11.4 Four stages of TMS320C54x

Table 11.2 Pipeline in different TMS320 Processors

DSP processor	**Pipeline phases**
TMS320C2000	F-D-R-X (4 levels)
TMS320C3x	F-D-R-X (4 levels)
TMS320C5x	F-D-R-X (4 levels)
TMS320C54x	PF-F-D-A-R-X (6 levels)

Pipelining leads to dramatic improvements in system performance. The more stages that we can break the pipeline into, the more theoretical speed we can get from it. For example, let's suppose it takes 12 clock cycles to handle all the steps to process an instruction. In theory, if you use a 4-stage pipeline, your maximum throughput is 1 instruction every 3 cycles. But if you use a 6-stage pipeline, maximum throughput is 1 instruction every 2 cycles.