3.5. INTRODUCTION TO RISC

RISC, or Reduced Instruction Set Computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures.

What are RISCs and why do we need them?

(*i*) RISC architectures represent an important innovation in the area of computer organization.

(*ii*) The RISC architecture is an attempt to produce more CPU power by simplifying the instruction set of the CPU.

(*iii*) The opposed trend to RISC is that of complex instruction set computers (CISC).

MIPS, and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC. Certain design features have been characteristic of most RISC processors.

(*i*) **One Cycle Execution Time.** RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU.

(*ii*) **Pipelining.** A technique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process instructions.

(*iii*) **Large Number of Registers.** The RISC design philosophy generally incorporates a larger number of registers to prevent in large amount of interactions with memory.

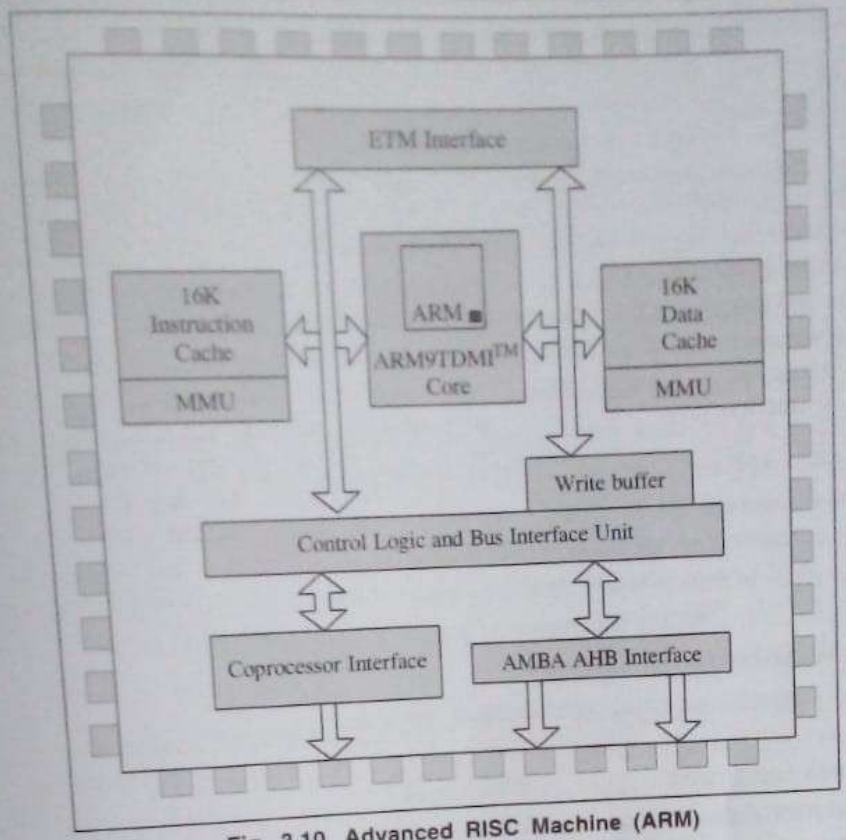
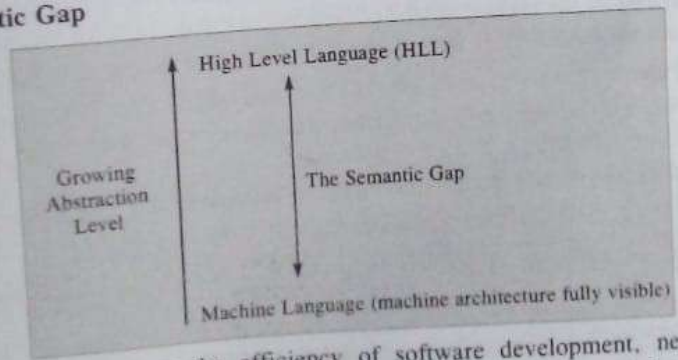


Fig. 3.10. Advanced RISC Machine (ARM)

Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap.

The Semantic Gap



(*i*) In order to improve the efficiency of software development, new and powerful programming languages have been developed (Ada, C++, and Java). They provide high level of abstraction, conciseness and power.

STORE R_2, A4

3.6. CISC (COMPLEX INSTRUCTION SET COMPUTER)

The design of an instruction set for a computer must take into consideration not only machine language constructs, but also the requirements imposed on the use of high-level programming languages. The translation from high-level to machine language programs is done by means of a compiler program. One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer

performance. The task of a compiler is to generate a sequence of machine instructions for each high-level language statement. The task is simplified if there are machine instructions that implement the statement directly. The essential goal of CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language. Another characteristic of CISC architecture is the incorporation of variable length instruction formats. Instructions that require register operands may be only two bytes in length, but instructions that need two memory addresses may need five bytes to include the entire instruction code, if the computer has 32 - bit words (four bytes), the first instruction occupies half a word, while the second instruction needs one word in addition to one byte in the next word. Packing variable instruction formats in a fixed - length memory word requires special decoding circuits that count bytes within words and frame the instruction according to their byte length. The instruction in a typical CISC processor provide direct manipulation of operands residing in memory.

Main characteristics of CISC Architectures :

1. A large number of instructions - typically from 100 to 250 instructions.
2. Some instruction that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes - typically from 5 to 20 different modes.
4. Variable - length instruction formats.
5. Instruction that manipulate operands in memory.

Some Processor Examples :

CISC Architectures:

1. VAX 11/780

No. of instructions	: 303
Instruction size	: 2 - 57
Instruction format	: not fixed
Addressing modes	: 22
Number of general purpose registers	: 16

2. Pentium

No. of instructions	: 235
Instruction size	: 1 - 11
Instruction format	: not fixed
Addressing modes	: 11
Number of general purpose registers	: 8

RISC Architectures :

1. Sun SPARC

No. of instructions	: 52
Instruction size	: 4
Instruction format	: fixed

Addressing modes	:	2
Number of general purpose registers	:	up to 520

2. PowerPC

No. of instructions	:	206
Instruction size	:	4
Instruction format	:	not fixed (but small differences)
Addressing modes	:	2
Number of general purpose registers	:	32

Summary

1. The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.
2. RISC stands for Reduced Instruction Set Computer.
3. CISC stands for Complex Instruction Set Computer.
4. MIMD stands for Multiple Instruction Stream, Multiple Data stream.
5. An operation code field specifies the operation to be performed.
6. RPN stands for Reverse Polish Notation.
7. The register set stores intermediate data used during the execution of the instructions.
8. Register and memory are types of stack.
9. Stack organization is very effective for evaluating arithmetic expressions.
10. LOAD instructions require memory access and their execution cannot be completed in a single clock cycle.
11. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
12. In the implied mode, the operands are specified implicitly in the definition of the instruction.
13. In register mode, the operands are in registers that reside within the CPU.
14. Two address instructions are the most common in commercial computers.
15. The PUSH and POP instructions, need an address field to specify the operand that communicates with the stack.

...nt and Expected Questions

CHAPTER 7

Overview of Advanced Microprocessor Technologies

7.1. PIPELINE PROCESSING

7.1.1. Parallel Processing

- A parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- The system may have two or more ALUs and be able to execute two or more instructions at the same time.

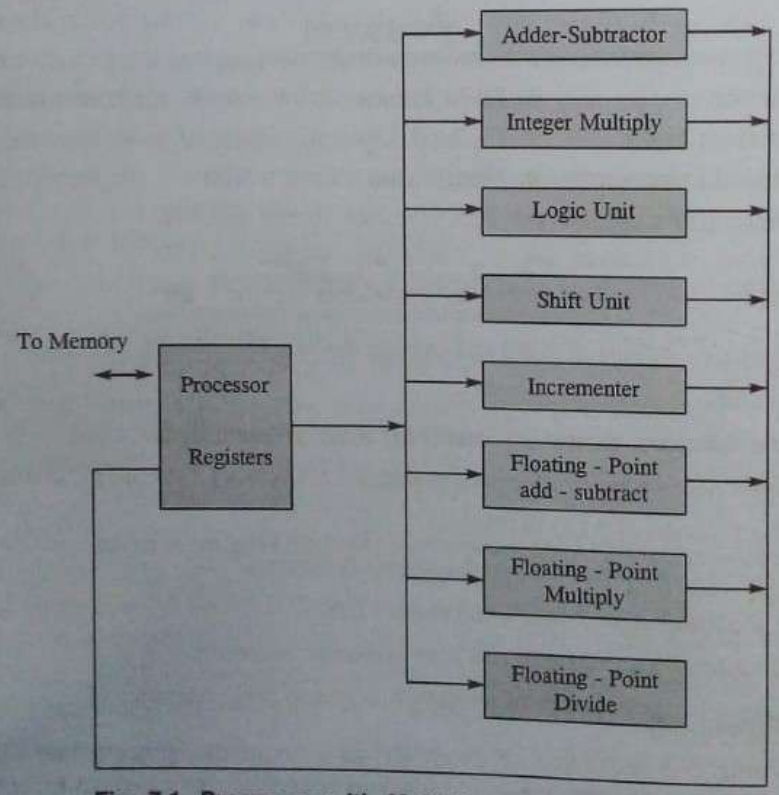


Fig. 7.1. Processor with Multiple Functional Units.

- Also, the system may have two or more processors operating concurrently.
- Goal is to increase the throughput – the amount of processing that can be accomplished during a given interval of time.
- Parallel processing increases the amount of hardware required.
- Example : The ALU can be separated into three units and the operands diverted to each unit under the supervision of a control unit.
- All units are independent of each other.
- A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.
- Parallel processing can be classified from:
  - The internal organization of the processors.
  - The interconnection structure between processors.
  - The flow of information through the system.
  - The number of instructions and data items that are manipulated simultaneously.
- The sequence of instructions read from memory is the instruction stream.
- The operations performed on the data in the processor is the data stream.
- Parallel processing may occur in the instruction stream, the data stream, or both.
- Computer classification :
  - Single instruction stream, single data stream – SISD.
  - Single instruction stream, multiple data stream – SIMD.
  - Multiple instruction stream, single data stream – MISD.
  - Multiple instruction stream, multiple data stream – MIMD.
- SISD – Instructions are executed sequentially. Parallel processing may be achieved by means of multiple functional units or by pipeline processing.
- SIMD – Includes multiple processing units with a single control unit. All processors receive the same instruction, but operate on different data.
- MIMD – A computer system capable of processing several programs at the same time.

We will consider parallel processing under the following main topics:

- Pipeline processing
- Vector processing
- Array processors

7.1.2. Pipelining

- Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- Each segment performs partial processing dictated by the way the task is partitioned.

- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- The final result is obtained after the data have passed through all segments.
- Pipelining can imagine that each segment consists of an input register followed by a combinational circuit.
- A clock is applied to all registers after enough time has elapsed to perform all segment activity.
- The information flows through the pipeline one step at a time.

Example :

$$A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7$$

The sub operations performed in each segment are:

$$R_1 \leftarrow A_i, R_2 \leftarrow B_i$$
$$R_3 \leftarrow R_1 * R_2, R_4 \leftarrow C_i$$
$$R_5 \leftarrow R_3 + R_4$$

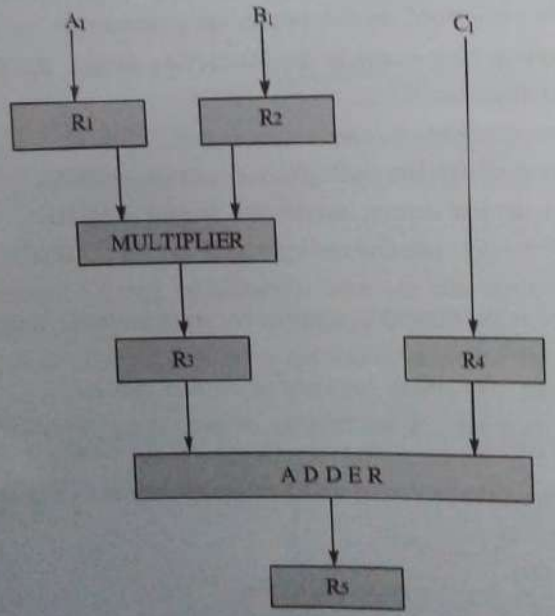
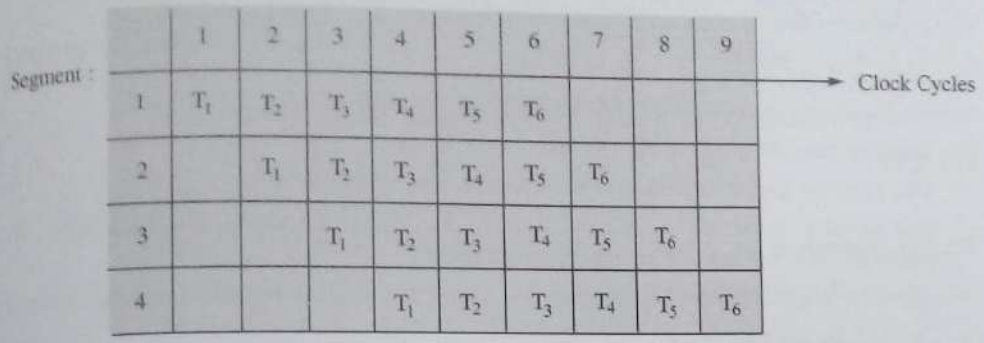


Fig. 7.2. Example of Pipeline Processing.

- Any operation that can be decomposed into a sequence of sub operations of about the same complexity can be implemented by a pipeline processor.
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data.

- A task is the total operation performed going through all segments of a pipeline.
- The behaviour of a pipeline can be illustrated with a *space-time* diagram.
- This shows the segment utilization as a function of time.
- Once the pipeline is full, it takes only one clock period to obtain an output.



	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

Fig. 7.3. Space – Time Diagram for Pipeline.

- Consider a k-segment pipeline with a clock cycle time t_P to execute n tasks.
- The first task T_1 requires time kt_P to complete
- The remaining $n - 1$ tasks finish at the rate of one task per clock cycle and will be completed after time $(n - 1)\ t_P$
- The total time to complete the n tasks is $[k + n - 1]\ t_P$
- The example of figure 7.3 requires [4 + 6 – 1] clock cycles to finish.
- Consider a nonpipeline unit that performs the same operation and takes t_n time to complete each task.
- The total time to complete n tasks would be nt_n.
- The *speedup* of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k+n-1)t_p}$$

- As the number of tasks increase, the speedup becomes

$$S = \frac{t_n}{t_p}$$

- If we assume that the time to process a task is the same in both circuits, $t_n = k\ t_P$

$$S = \frac{t_n}{t_p} = k$$

- Therefore, the theoretical maximum speed up that a pipeline can provide is k.