Embedded System Design with ARM

Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 04
Architecture of ARM Microcontroller (Part I)

In this lecture we shall be starting our discussion on something about the ARM microcontrollers. What are their architecture like, what are their specific features like and how are they different from the earlier generation of microcontrollers ok. So, the topic of this lecture is Architecture of ARM Microcontroller, this is the first part of the lecture.

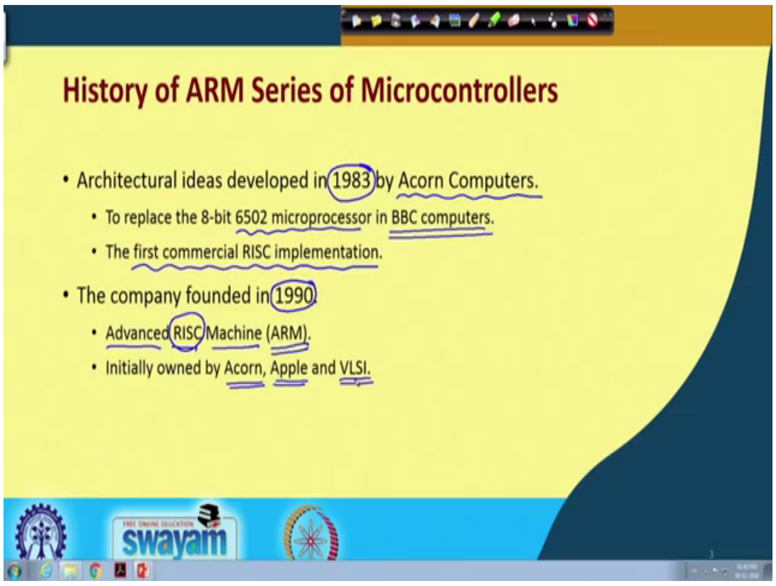
(Refer Slide Time: 00:41)

CONCEPTS COVERED

- ARM series of microcontrollers
- Evolution of ARM microcontrollers
- Important ARM features

Now, in this lecture we shall be covering some general ideas about the ARM series of microcontrollers, how they have evolved and some of the important architectural features ok.

(Refer Slide Time: 00:57)



History of ARM Series of Microcontrollers

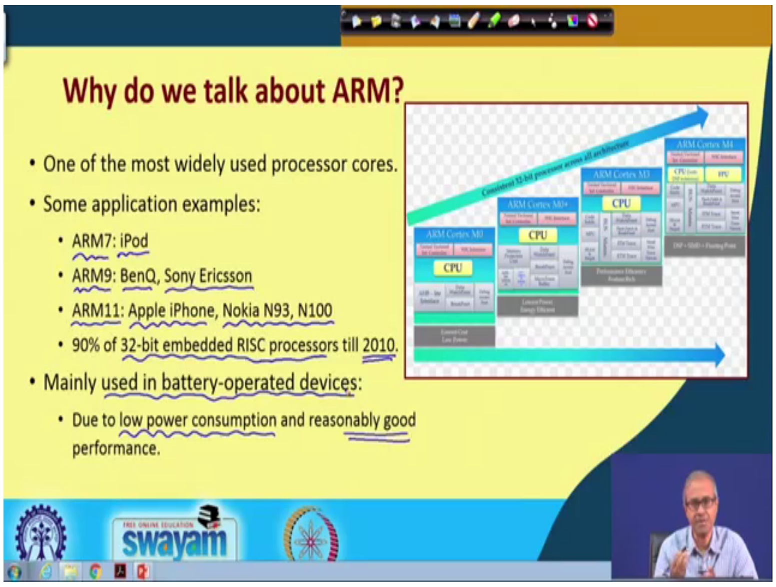
- Architectural ideas developed in 1983 by Acorn Computers.
  - To replace the 8-bit 6502 microprocessor in BBC computers.
  - The first commercial RISC implementation.
- The company founded in 1990.
  - Advanced RISC Machine (ARM).
  - Initially owned by Acorn, Apple and VLSI.

Let us look into the history first very briefly. Now the architectural ideas that built or that have evolved into this ARM class of microcontrollers. This was developed long back in 1983. There was a company called Acorn computers which was the first to develop and evolve such ideas. Now these ideas were little different because they started to develop the architectural ideas based on the reduced instruction set concept, RISC architecture concept ok. And at that time there was a very popular microprocessor called 6502 from a company called M~~m~~ostech that was used in one of the very popular microcomputers called BBC micro that was a microcomputer which was used quite widely and during that time and this M~~m~~ostech 6502 was the microprocessor at that time it was an 8 bit microprocessor that was inside that microcomputer.

So, the first attempt of these people were to replace that processor by a better processor more powerful processor, which will make the BBC micro faster and more powerful ok. So, this resulted in the first commercial RISC implementation. So, it was not called armed during that time, but it evolved into the ARM architecture. There was a company which finally, got founded in 1990. The name ARM is the acronym for advanced RISC machine. So, you see in the name itself the word RISC is embedded.

So, ARM architecture essentially borrows the concepts from the RISC idea, from the RISC architectural concept ok. And initially this company ARM was jointly formed and owned by this account which was the initiator, Apple was also there and there was another company called VLSI. This 3 companies came together and formed this new company called ARM..

(Refer Slide Time: 03:37)



Why do we talk about ARM?

- One of the most widely used processor cores.
- Some application examples:
  - ARM7: iPod
  - ARM9: BenQ, Sony Ericsson
  - ARM11: Apple iPhone, Nokia N93, N100
  - 90% of 32-bit embedded RISC processors till 2010.
- Mainly used in battery-operated devices:
  - Due to low power consumption and reasonably good performance.

Now what is so interesting about ARM? So, why do you have to talk specifically about ARM? You may ask in this course why are you specifically trying to use ARM as the vehicle for teaching embedded systems. The reason is this ARM have been this has been you can say increasingly used in many applications, they are the most popular category of microcontrollers which has seriously used in embedded system applications.

Let us take some examples that you all know about the iPods from Apple through which you can listen to music, you used to listen to music now they are discontinued there was an ARM processor inside that. Benq, Sony Ericsson these are very well known companies they manufacture TV sets and many audio visual other equipments, there are ARM processors inside each of these equipments. Typically they started to use ARM 9, but subsequently they updated or upgraded them to the later version of ARM processors.

This Apple iPhone all of us are familiar with and some of the very popular Nokia phones. They all have the ARM11 processor inside them.

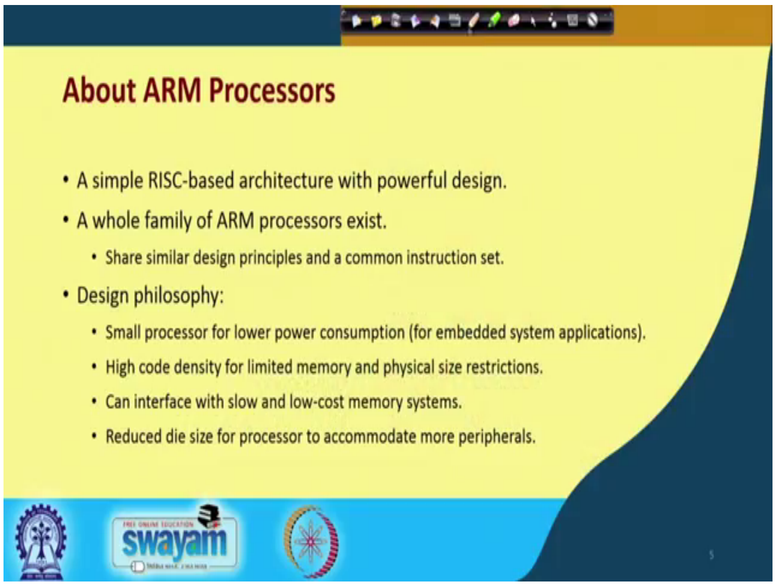
Now, this is a pretty old piece of statistics. Till 2010, 90 percent of all serious embedded applications has this kind of ARM processors inside them. Now you see let us talk about one thing, well when you talk about embedded system application there is a processor inside, I said depending on the application you decide how much power you need from the processor. If it is a very simple application you do not need a processor as powerful as ARM. The PIC I told you about there are very simple PIC processors available, which are very very cheap, those are very simple 8 bit processors you can even use them.

But ARM processors are typically 32 bit and above. So, you use ARM processor when you need reasonably powerful computation capability that will make the heart of your embedded system right. Now another thing is that this ARM processors they have very low power consumption and of course, reasonably good performance needless to say because of this low power consumption, they are very widely used in battery operated devices. There are many battery operated devices like the mobile phones; I talked about the iPods there were Apple processors inside ok.

And if you look at this diagram, this diagram is not really but just I wanted to show that over the years from left to right this is the axis of time, the armed 32 bit processors I was talking about in the last lecture that the present trend is to develop an application specific integrated circuit were not only the ARM processor but many other things are inside the chip.

So, you see here this boxes that we show here these are those ASIX, here CPU is only one part of it, there are many other things. Now across the years ARM Cortex M 0 M 0 plus M 3 M 4 these have evolved. Now the board with which you shall be giving some of the demonstrations this is actually based on the ARM Cortex M 4; one of the latest chips in the series right ok. Now talking about the ARM processors there are some very interesting features, let us look at some of the notable features one by one.

(Refer Slide Time: 08:09)



About ARM Processors

- A simple RISC-based architecture with powerful design.
- A whole family of ARM processors exist.
  - Share similar design principles and a common instruction set.
- Design philosophy:
  - Small processor for lower power consumption (for embedded system applications).
  - High code density for limited memory and physical size restrictions.
  - Can interface with slow and low-cost memory systems.
  - Reduced die size for processor to accommodate more peripherals.

First thing is that, it is essentially a reduced instruction set computer based architecture. Now we shall go into some detail what a RISC architecture is and the design is pretty powerful. It borrows some advanced architectural ideas in contrast to conventional or contemporary microcontrollers that had very primitive kind of designs.

Well I talked about 8051 that was a very popular microcontroller no doubt, but architectural wise it was pretty primitive, it did not use any kind of architectural enhancement or advanced features ok. Now as I told you this ARM processor is not just one, but a whole family of ARM processors exist and the most important thing is that in order to maintain backward compatibility, which is very important in this kind of evolution all of this share essentially a common instruction set.

Of course in the later generations some additional instructions have been added, but the older instructions are also carried through such that; a program which was developed for a older generation would run pretty well for the next generation also right. Now the design philosophy here was of course, first thing is that we need a small processor so that we can have lower power consumption and can be used for embedded systems application.

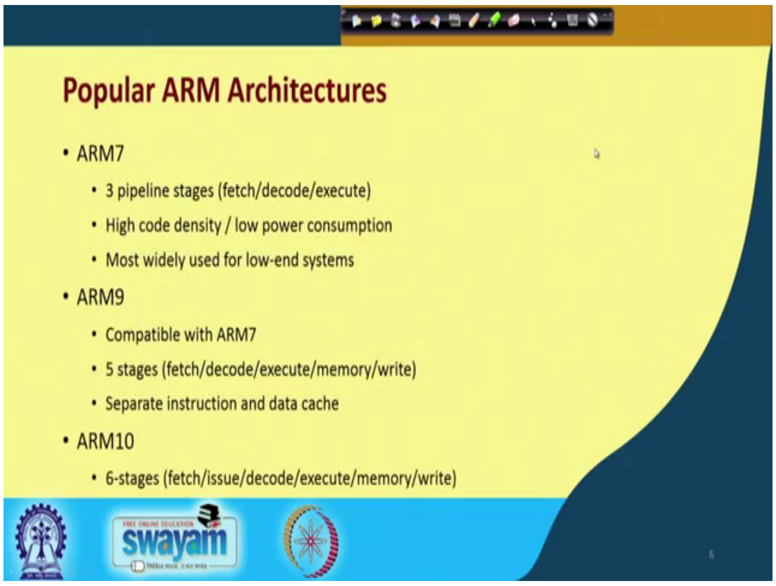
So, the size of the processor should be small this is one thing that is very important it should consume low power that is also very important right. And high code density what do mean by high code density? You see in microcontrollers I told that memory what program memory and data memory are all inside the chip. So, there is a scarcity of you can say real estate you cannot put very large memory inside.

So, there is a maximum limit to the size of the program that you can run. Let us say my program memory size is 100 kilobytes let us take an example 100 kilobytes. So, whatever I write must fit within this 100 kilobytes. So, if my instruction set supports that in this 100 kilobytes, I can pack my code very nicely, so that I can implement more functionality greater functionality as compared with some kind of competing architecture where a much more memory would be required to implement the same thing. What I mean to say is that, suppose I have an application x that I want to implement in a conventional architecture maybe it will be requiring 120 kilobytes but in RISC ARM Architecture I can fit it within 100 kilobytes.

This is something called high code density. There are some instruction features we shall be talking about which allows us to reduce the number of instructions required. So, this can take advantage of limited memory and physical size restrictions. And of course, here there is lot of flexibility in the interface. We can interface with a wide variety of memory systems, very slow or also relatively fast memory systems it can all be interfaced with depending on the scenario. And of course, reduced die size means when you are actually fabricating the chip. The size of that silicon is very small, so that when you develop that ASIC I was talking about that ARM would occupy a very small portion of it.

So, you can use the remaining space to put in much more you can say additional functionality to make the ASIC very powerful right ok.

(Refer Slide Time: 12:40)


Popular ARM Architectures

- ARM7
  - 3 pipeline stages (fetch/decode/execute)
  - High code density / low power consumption
  - Most widely used for low-end systems
- ARM9
  - Compatible with ARM7
  - 5 stages (fetch/decode/execute/memory/write)
  - Separate instruction and data cache
- ARM10
  - 6-stages (fetch/issue/decode/execute/memory/write)

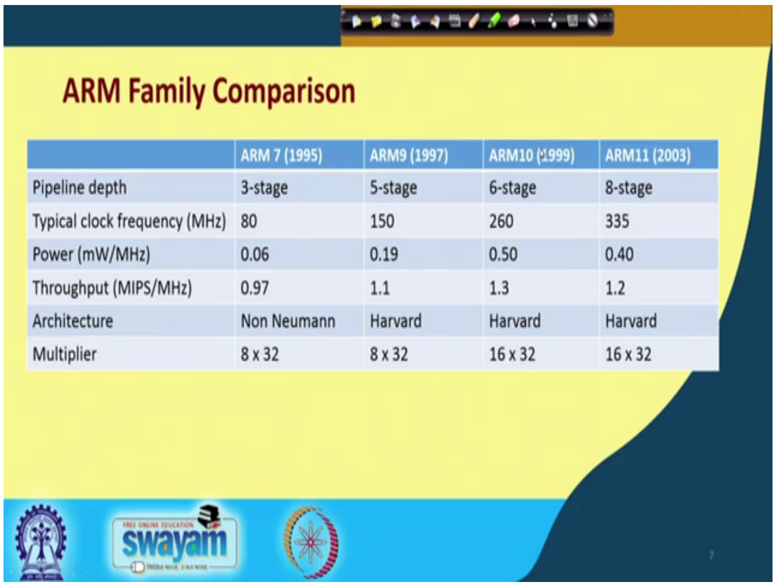
So, some of the popular ARM architecture there are many I am only shown 3, this ARM 7, 9, 10 there are 11 and beyond. Some of the main features I am written down, ARM 7 has 3 pipeline stages, we shall be talking about pipeline later in the next lecture. Now pipeline stage essentially means how the instructions get executed, there is a concept of pipelining we shall be explaining this.

So, there are 3 stages fetch decode execute. So, it supports high code density I told you low power consumption and for low end systems where very high power is not required this ARM 7 is a very popular architecture. So, you do not need very high power ARM processors everywhere whatever you need inside a mobile phone you will not need possibly inside a refrigerator, you need very simple kind of calculations there right. Coming to ARM 9 first thing is that these are all backward compatible, but the pipeline stages announced to 5 stages; fetch decode execute memory right and the concept of cache memory came in and there is a separate instruction cache and separate data cache

In ARM 7 instruction and data were both in the same memory. So, it was like a von Neumann architecture, but from ARM 9 onwards the architecture became it started a shift towards Harvard architecture right. Talking to ARM 10, the main difference was the pipeline was further enhanced by adding another stage this issue, this issue was added to this right.

So, in this way the basic architecture started evolving making the processor more powerful and faster by adding novel architectural concepts.

(Refer Slide Time: 14:42)



ARM Family Comparison

	ARM 7 (1995)	ARM9 (1997)	ARM10 (1999)	ARM11 (2003)
Pipeline depth	3-stage	5-stage	6-stage	8-stage
Typical clock frequency (MHz)	80	150	260	335
Power (mW/MHz)	0.06	0.19	0.50	0.40
Throughput (MIPS/MHz)	0.97	1.1	1.3	1.2
Architecture	Non Neumann	Harvard	Harvard	Harvard
Multiplier	8 x 32	8 x 32	16 x 32	16 x 32

Now, this table gives a quick comparison among 4 ARM family members ARM 7, 9, 10 and 11. Well you can also see the year when it was first introduced 95, 97, 99 and 2003. First thing is pipeline depth; depth means how many stages of the pipeline are there? We only we already talked about ARM 3 stage, ARM 9 5 stage, ARM 10 6 stage and then ARM 11 we have 8 stages.

So, we are enhancing the number of stages in the pipeline to make the execution faster in some sense. The clock frequency sometimes the speed of a processor is determined by how fast we can make a clock. In ARM 7 it was 80 megahertz then 150, 260, 335 and so on. So, the clock frequencies are increasing, the processors are becoming faster. Power, power if you see the power consumption is a measure of the clock frequency, faster is the clock more will be the power consumption.

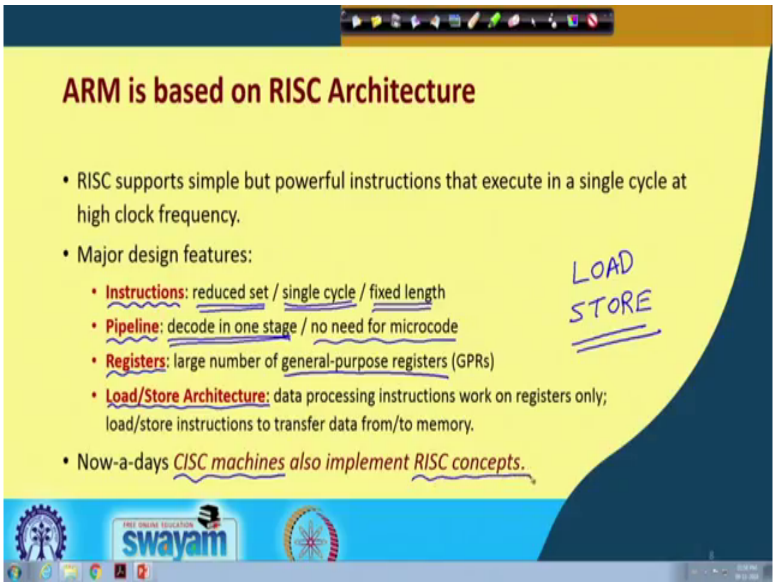
So, you should estimate the power consumption with respect to the clock frequency we are using because every microcontroller has a range of permissible clock frequencies. So, it depends upon you what clock frequency do you want, if you can operate with a lower clock frequency and serve your purpose it is fine you will be you will be consuming much lower power. So, power consumption in microcontrollers are typically measured by milliwatt per megahertz ok.

So, higher the megahertz clock you just multiply it by that you will get the total mini watt of power consumption. So, for ARM 7 it was 0.06, 0.19, 0.50 and 0.40. You see in ARM 11 due to some low power design take makes the power consumption got reduced from 0.5 to 0.4 and throughput is how fast instructions can get executed. Now again throughput is a function of the clock frequency.

So, in microcontrollers again you can measure throughput typically by million instructions per second per megahertz. So, ~~so~~ if it is on 80 mega you have to multiply this by 80 you will get so many million instructions per second. So, the figures are 0.97, 1.1, 1.3, 1.2. Architecture as it said ARM 7 was based on von Neumann this is v von Neumann, but subsequently there is a move towards Harvard Architectures and inside the processor there is a built in multiplier.

So, there was an 8 by 32 multiply in the first 2 generations whereas, for the next two generation there was a 16 by 32 multiplier because many instructions, many applications for example, the digital signal processing applications they frequently require multiplication operation. So, if there is a hardware multiplier built in it speeds up operation quite significantly.

(Refer Slide Time: 18:13)



ARM is based on RISC Architecture

- RISC supports simple but powerful instructions that execute in a single cycle at high clock frequency.
- Major design features:
  - Instructions: reduced set / single cycle / fixed length
  - Pipeline: decode in one stage / no need for microcode
  - Registers: large number of general-purpose registers (GPRs)
  - Load/Store Architecture: data processing instructions work on registers only; load/store instructions to transfer data from/to memory.
- Now-a-days *CISC machines also implement RISC concepts.*

LOAD STORE

Now, the point to note I said that ARM the basic concept or evolution started from the RISC architecture. So, ARM is based on the RISC architecture. So, what is RISC? RISC is based on some architectural features.

This architectural features are like this, with respect to instructions. There is less number of instructions reduced set. Instructions are simple so that all instructions can be executed in a single cycle. They are all of fixed length so that decoding of the instruction becomes very easy and your hardware for the controller becomes very simple ok. Then with respect to the pipeline here we shall see later that instructions are typically executed in a pipeline in all modern day processors. Now if the instructions are simple they are fixed length, then decoding of the instruction becomes very easy. You can decode in one stage itself, you do not have to again look at the instruction and try to find out what this instruction was ok.

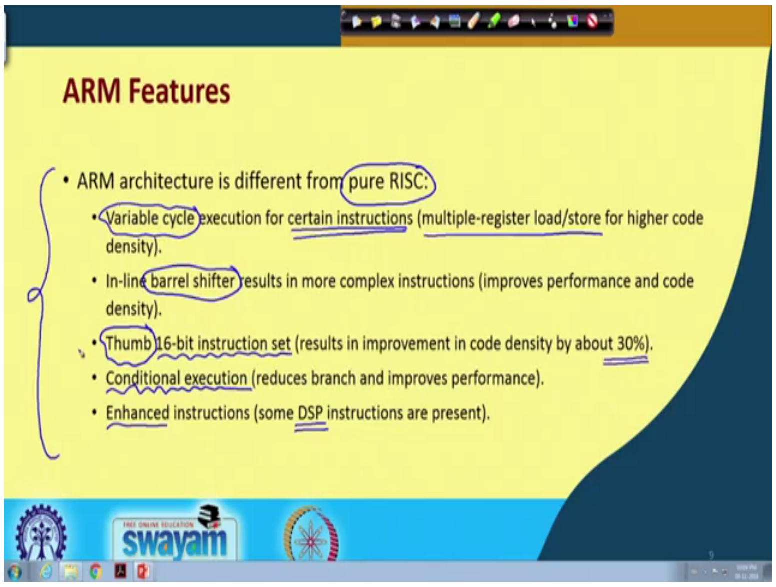
So, there is no need for micro programming which is a standard norm for the complex instruction set computers. You can directly implement the control unit in hardware, if you do it in hardware it also becomes much faster you can run it at a higher clock. Registers one characteristic of RISC architecture is that there is a very large number of general purpose registers, typically 32 or more there are a large number registered where you can temporarily store your data during calculations.

There are very few special purpose registers unlike CISC Architectures where there are lot of special purpose registers like program counters stack pointer base registers and so on and so forth right. And another important thing is that RISC is based on a load store architecture means there are some load and store instructions load and store these instructions are responsible for transferring data between registers and memory. All other instructions the ALU instructions arithmetic and logic instructions they only work on registers, they do not access memory.

These kind of instruction set is sometimes called load store architecture, that only load and store instructions access memory all other instructions they work only on the registers right. Now I told earlier that even the CISC machines of today the Intel class of machines they use micro programming, they translate those complex instructions into some kind of micro programs simpler instructions which look more like the RISC instructions.

So, they also implement RISC concepts in some way, they make an initial translation which they execute using standard RISC techniques, RISC instruction execution techniques right. So, these are some of the concepts behind the RISC architecture. Now talking about ARM, well although the name ARM contained this RISC this middle R is the acronym for RISC.

(Refer Slide Time: 22:13)



ARM Features

- ARM architecture is different from pure RISC:
  - Variable cycle execution for certain instructions (multiple-register load/store for higher code density).
  - In-line barrel shifter results in more complex instructions (improves performance and code density).
  - Thumb 16-bit instruction set (results in improvement in code density by about 30%).
  - Conditional execution (reduces branch and improves performance).
  - Enhanced instructions (some DSP instructions are present).

But strictly speaking ARM is not a pure RISC Architecture; there are some features which have been introduced in the architecture because they are very useful in embedded system applications, which are not RISC characteristics. So, it is their ARM starts to deviate slightly from the RISC architectural concepts. Some of the differences are as follows: Certain instructions require variable number of clock cycles for execution. So, while talking of RISC I said all instructions should be exhibited in a single clock cycle, but in ARM some of the instructions can be more complex, it can require multiple clocks such instructions are there.

So, one classical example is multiple register load store. Like normally we will load a value from memory into 1 register, but ARM allows you to specify in such a way that the value loaded will be loaded into let us say 4 registers. So, to write into those 4 registers you need 4 clock pulses. So, you cannot do it in one cycle you need multiple cycles right, such multiple data transfer instructions are supported in all. And there is something called a barrel shifter which is a very common architectural concept.

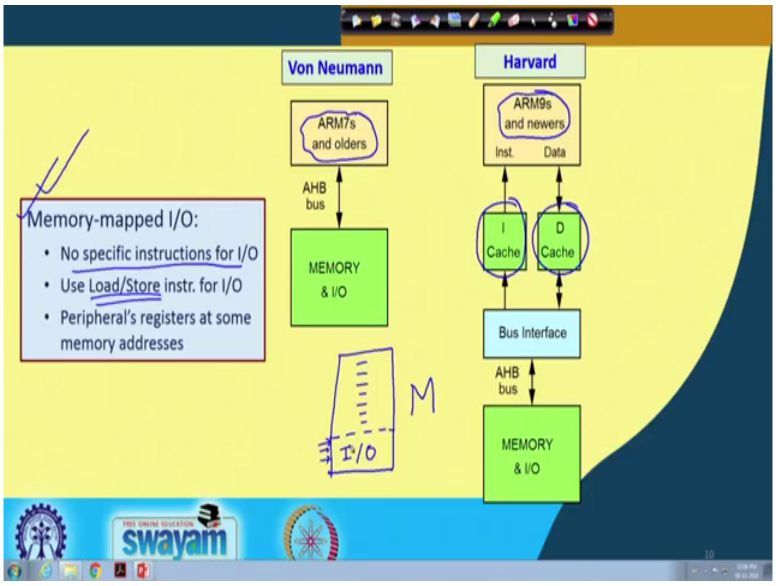
Barrel shifter is a hardware which allows multiple bit shifting very efficiently in a single cycle. So, this barrel shifter is part of the ARM architecture and there are many instructions which directly utilize this barrel shifting capability. Let me take an example suppose there is an add instruction which adds 2 registers let us say r 2 and r 3 it adds, but I can also say you add r 2 and r 3 shifted left by 4 positions. So shifted left by 4 positions will be done by the barrel shifter, it will not take any additional time, in that single clock cycle everything can be done.

Because of the presence of the barrel shifter this kind of shift and operate kind of instructions are possible to implement in a very efficient way. And another feature is that there is a feature in ARM instruction set in ~~in~~ the ARM architecture you can say, that you can configure it in the thumb mode. Thumb means thumb is a subset of the of the ARM instructions, which works in 16 bit mode. Normally ARM processors are 32 bit processors, but there may be many application where you do not need that power, you need much simpler power you can have the thumb instruction set which is essentially a 16 bit instruction set right so.

So, if we use instructions which are smaller this can further lead to a shortening of the total code size your code density can further improve right. And there is another very interesting feature we shall be discussing this in detail conditional execution. Like you can say you add these 2 numbers provided the 0 flag is set. Well in conventional processors if the 0 flag is set you can have a jump if 0 jump if non ~~0~~ zero kind of instructions, you do a jump then check if it is not 0, then add a draw is do not add; that means, you need so many jump instructions.

But if you have a conditional instruction like you say add if 0 flag is set then you are avoiding the jump instruction altogether. So, number of instructions, also get reduced right. And of course, there are some enhancement, some digital signal processing instructions like one example is multiply and add, this kind of instructions are there they have been added to the instruction set. Because of these so ARM has deviated slightly from the pure RISC you can say category, but still it is a fairly powerful processor mostly based on ~~rise~~RISC, only for a few cases it deviates because of very good reasons of course.

(Refer Slide Time: 27:14)


Memory-mapped I/O:

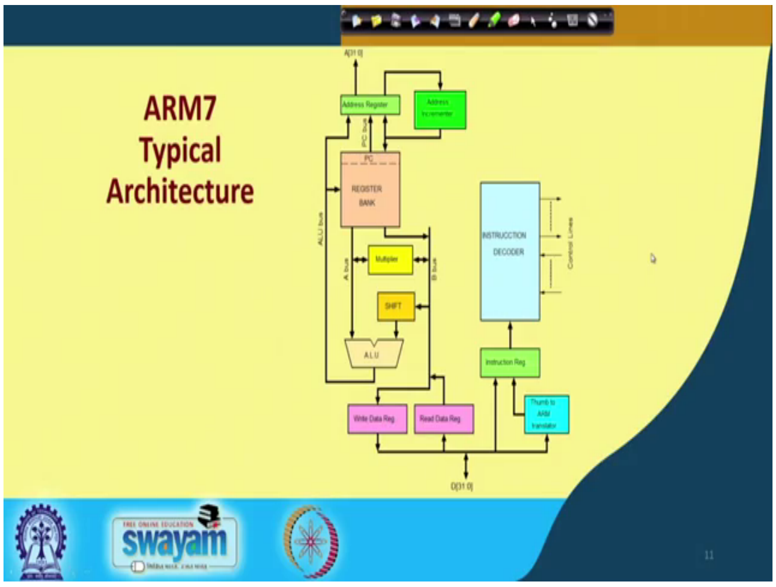
- No specific instructions for I/O
- Use Load/Store instr. for I/O
- Peripheral's registers at some memory addresses

Now talking about this von Neumann and Harvard architecture you already talked about earlier, this ARM 7 and the even older processors were based on von Neumann, there was a single memory and r ~~nine~~ 9 and the later processors they have 2 separate memory instruction memory and data memory and inside also there is an instruction cache and a data cache separate. Now another feature is that I would like to say is that ARM processors does not have any separate instructions for input output, they use something called memory mapped IO.

Like let us say this is your total memory area, this is your memory right? Now there is 1 part of memory which you reserved for the IO devices. Normally when you access memory you store the data here, but when you are trying to access some address in this region there were decoding circuitry which will automatically be accessing the IO ports instead of the memory.

But there will be a same decoder for memory and IO operation. Say load store instructions are typically used to transfer data between memory and register, the same load store instructions will be used for reading from IO port or writing into IO ports, there are no separate instructions for input and output ok. So, no specific instructions for IO, you use the same load store instructions right. The address to be used will be the address corresponding to the IO devices. Well we are not going into the detail of this just the basic concepts.

(Refer Slide Time: 29:08)



ARM7 Typical Architecture

So, this is the ARM typical architecture, I just wanted to show you just a snapshot of it. You see you have all the registers say register bank, here we have the arithmetic logic unit. So, one of the data is coming directly from the register bank here and the other one you see there is a barrel shifter sitting here.

So, if the other data can be shifted and then apply to ALU or it can even come without that, so with no shifting. The multiplier is sitting here whenever you need this multiplier hardware you can multiply and bring it to the, a bus and there are some other instruction features. There is an address register address implementer, so these are the address bus and here is the data bus for interfacing with memory.

But inside this is interesting, there is a multiplier, there is a barrel shifter which are sitting before the ALU. This makes the implementation of some of the instructions very efficient. So, with this we come to the end of this lecture. We shall be continuing this discussion over the next couple of lectures where we shall be looking at some of the more additional features that are there in the ARM instruction set and also the ARM architectures.

Thank you.