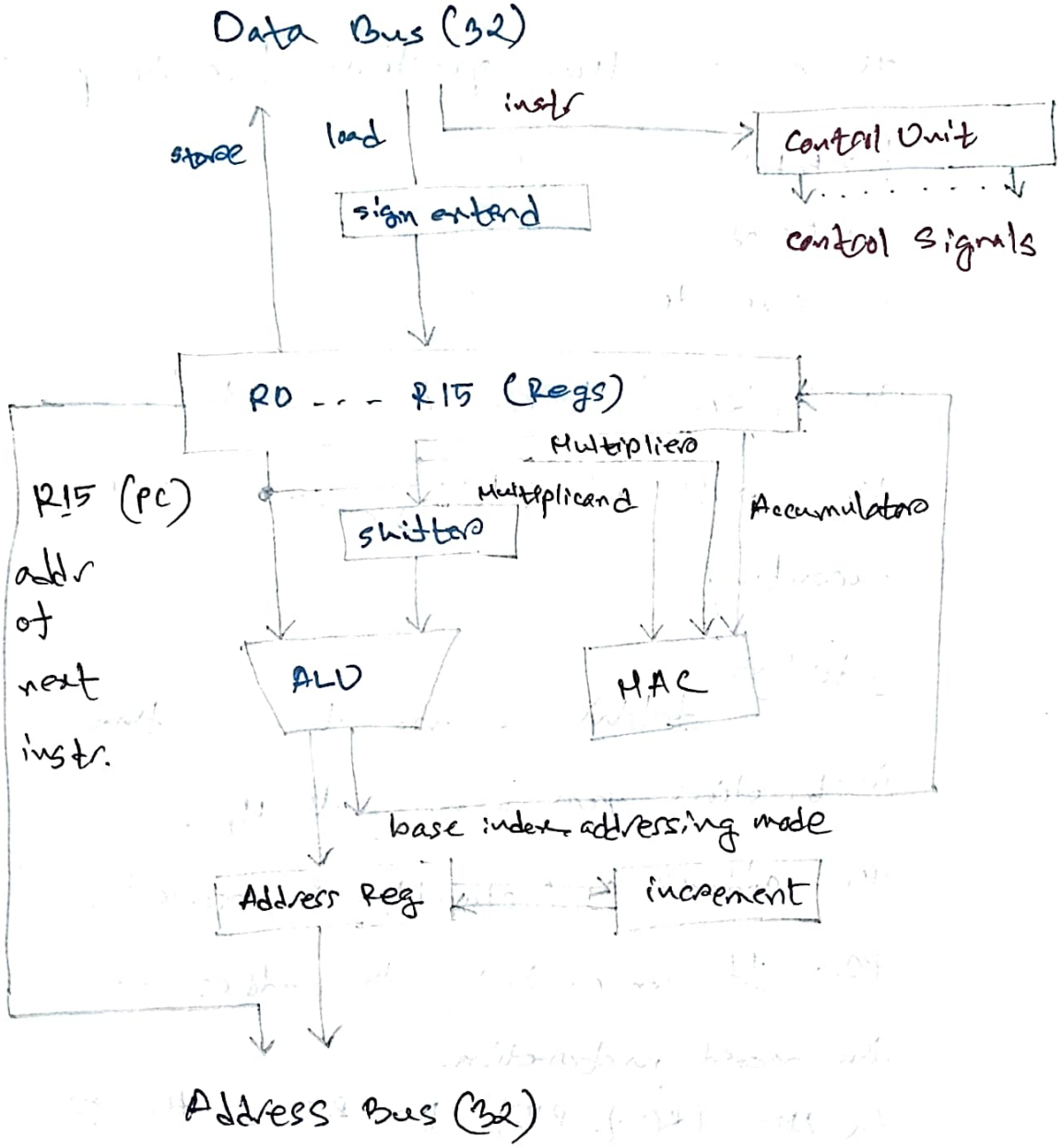
ARM7 Data flow Model, Core Architecture



How to learn an architecture of a processor?

There are three oparitions done in processor.

- Fetch an instr
- Decode
- Execute.

The system of learning it is to trace the path of fetching, decoding and executing.

Fetching:-

fetching means transfer the instruction from memory to Mp.

The address of instruction is stored in PC. It can containg the addres of the next instruction.

In ARM (R0 to R15) the R15 is the PC registers.

The PC will goto on the address bus (32).

The address bus will go to the location given by the pc. In that location the data is stored. This data can be is

transfered to the data bus (32).

This data can be instruction or normal data.

If it is an instruction data bus sends the information to the "control unit".

If ps is normal data data bus sends the information to the "sign Extn".

Decoding:- The section in architecture decodes the instruction is called "control unit"

There are two types of control unit.

1. Hardwired → Rigid but very fast
2. Microprogram → flexible but slow.

As ARM is a performance based μp so its control unit is hardwired based.

Decoding takes place exactly in one # cycle

Decoding means understanding the opco opcode what has to be done. After decoding 'Control Unit' sends signals to the total ~~system of control~~ system of the architecture to operate instruction.

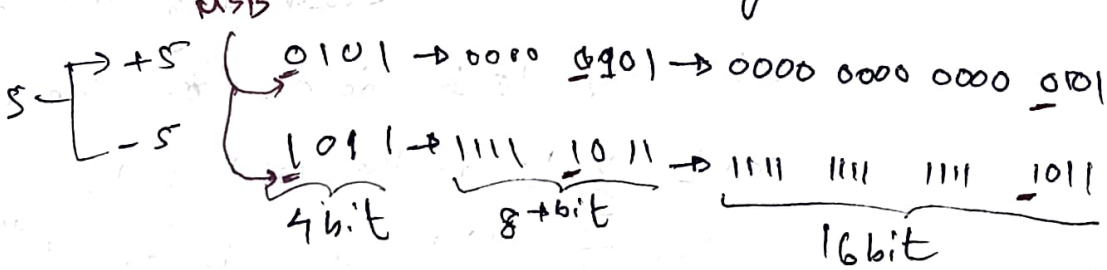
Execution:

In ARM all the operations are done through registers. Only operation that is done on memory is Load and Store. When any value comes through databus, if the value is an instruction it goes 'control unit' and if the value is a data it goes to 'Signal Extend' unit.

what is the requirement of sign extender?

When we load any data from memory the data can be 8bit, 16 bit or 32 bit.
The reg register where the data will be kept is 32bit. If the data is 32bit it directly goes to the register. If the data is 16 bit or 8 bit, it has to be extended and make it 32 bit then sends it to registers.

The section which converts 8bit or 16 bit data to 32 bit is called 'Sign Extender.'#



The data is extended to 32 bit by replicating using the MSB of the number. In this way the actual value of the data remains same. But at the same time it becomes 8 bit to 32 bit. or 16 bit to 32 bit.

Another question arises there is system for storing 8 bit or 16 bit data. for extension in store the data. The value in Register is always 32 bit. So there is no requirement for extension of data.

Triadic Instrs:

ARM supports triadic instr. Ex: ADD R0, R1, R2

it means R0 = R1 + R2

R0 register will store the sum of R1 and R2

It is not possible to add 3 register at a single operation. Because in arithmetic operation if overflow occurs it stores the extra bit in carry flag. The carry bit can be 0 or 1

It can not be 2, 3 or more.

So adding two register maximum overflow is 1.

$$\begin{array}{r} 9 \\ +9 \\ \hline (1)8 \end{array} \qquad \begin{array}{r} {}^{1\,1}\ \ \\ 99 \\ 99 \\ \hline (1)98 \end{array}$$

If we add three register

$$\begin{array}{r} 9 \\ 9 \\ 9 \\ \hline (2)7 \end{array} \qquad \begin{array}{r} 99 \\ 99 \\ 99 \\ \hline (2)97 \end{array}$$

So here the overflow bit is 2.

for carry flag it is not possible to ~~keep~~ store '2' in a single bit.

So in processor summation of three register at a time is not possible.

The operation can be,

ADD R1, R1, R2 means R1 = R1+R2

ADD R0, R0, R0 means R0 = R0+R0.

but below one is not possible,

R0 = R1+R2+R3 (x).

shifter: (officially it is written as barrel shifter)

all modern shifters are barrel shifters. It can shift all in one cycle. It is used in addressing modes.

It is used in based indexed scaling.

Multiplied by 2 shifts leftward. one bit.

division by 2 shifts rightward by one bit.

ALU:

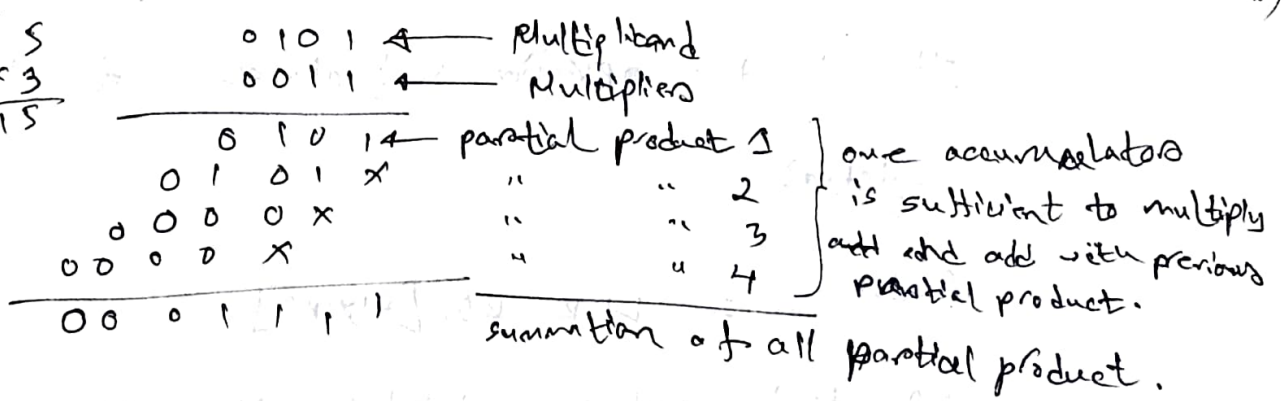
ALU performs the operation and stores the result in the register.

MAC:

First understand what is MLA. (Multiply and Accumulate).

It is an instruction. MLA R0, R1, R2

This MAC is used when we do bigger multication (64bit x 64bit)



when we do multiplication in papers there is one no. of multipliers = no. of partial product.

In any processor if we want to do the same thing. we will require 64 registers for storing 64 partial product in 64 bit multipliers. But it is not possible.

But if we use one accumulators to storing all the partial products and add them with the previous one, the ultimate result will remain the same.

multiplication

So any bigger operation there requires three things multiplicand, multiplier and accumulator. Thats the thing is done by MAC.

It do the operation and stores it into the register.

Address Reg:- LDR R0, [R1, R2]

R0 gets the address pointed by R1 and R2.

In "base index addressing mode" the one register gives the base of the address and the other register gives the index of the address.

Suppose we want to access 2004, 2002, 2009, 2001 locations frequently then we initialise one register (R1) with base value then change the index value (R2) where we want to access.

The base value 2 becomes 2000 and 1 remains 4.

In (2 to 2000) the shifter makes 2 into 2000.

Finally ALU calculates the address for any particular location.

These is an interesting thing comes, if we want to access location. 2000, 2001, 2002, 2003... sequentially one by one.

In ARM there is an "incrementer" to access a series of locations one by one.

When this incrementer is used to get next address ALU is not used to generate sequencial address. Means ALU does not get disturbed. By this time the ALU can do others operations.

If we want to access series of locations ALU generate the first location and gives it to

'Address Reg' - And the further addresses are generated by 'increment' registers.

Division 15 ÷ 3 = 5 quotient is 5, remainder 0.

15 ÷ 6 = 2.5...

~~divisor~~

ARM7 tdmi ~~does~~ can not operate with ~~operation~~ the floating point numbers.

It can do division without floating points.

If we want to ~~use~~ open ~~+~~ perform floating point operation we have to use the processors with it.