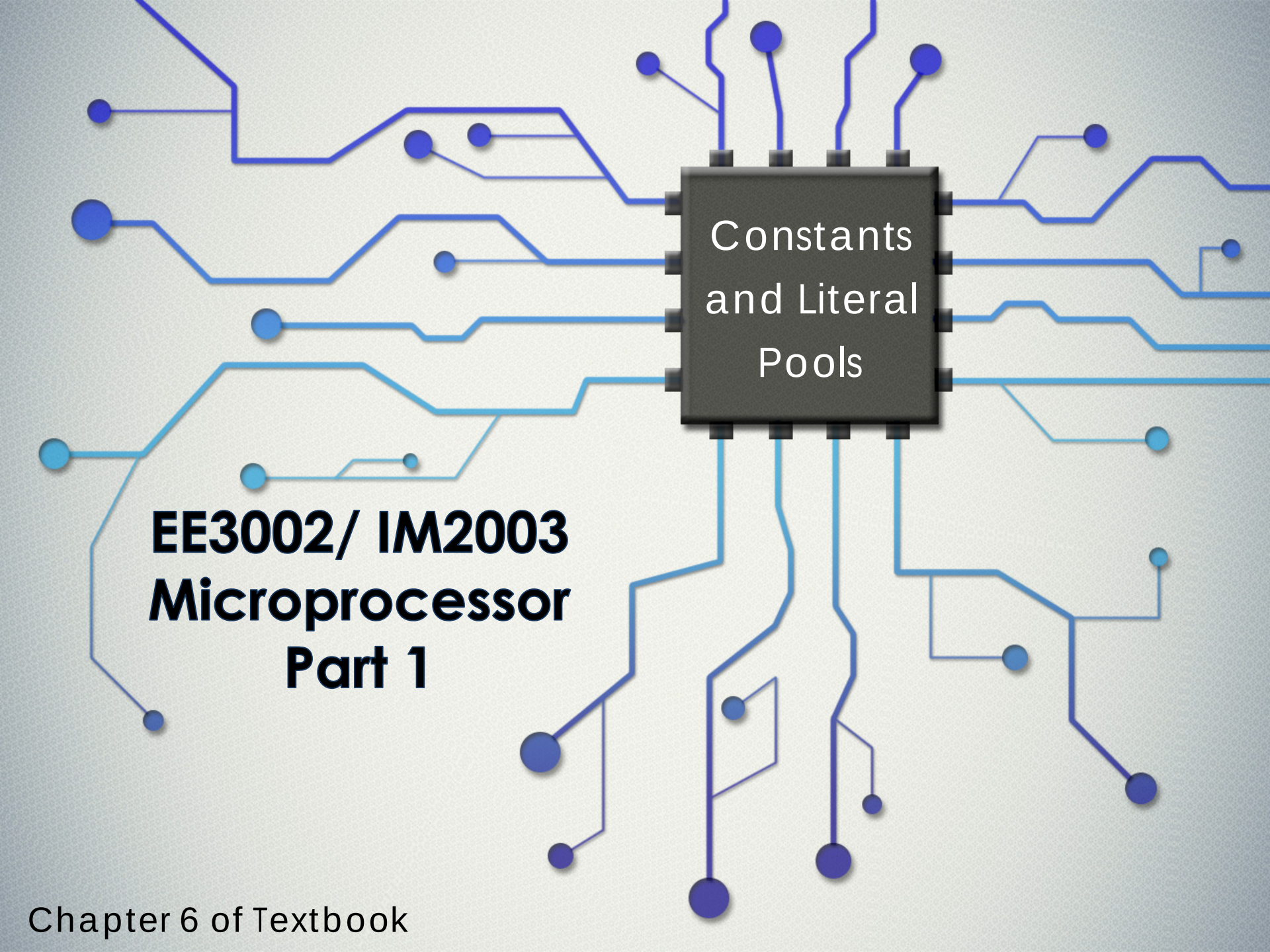


EE3002/ IM2003 Microprocessor Part 1

Chapter 6 of Textbook

ARM Rotation Scheme

- ARM instructions are 32 bits in length
- How do you load a 32-bit constant into an instruction that is only 32 bits long?
- MOV instruction

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction Type
Condition				0	0	0	1	1	0	1	S	0	0	0	0	Rd				Shifter_operand												MOV
							OPCODE					Rn																				

- MOV instruction with immediate operand

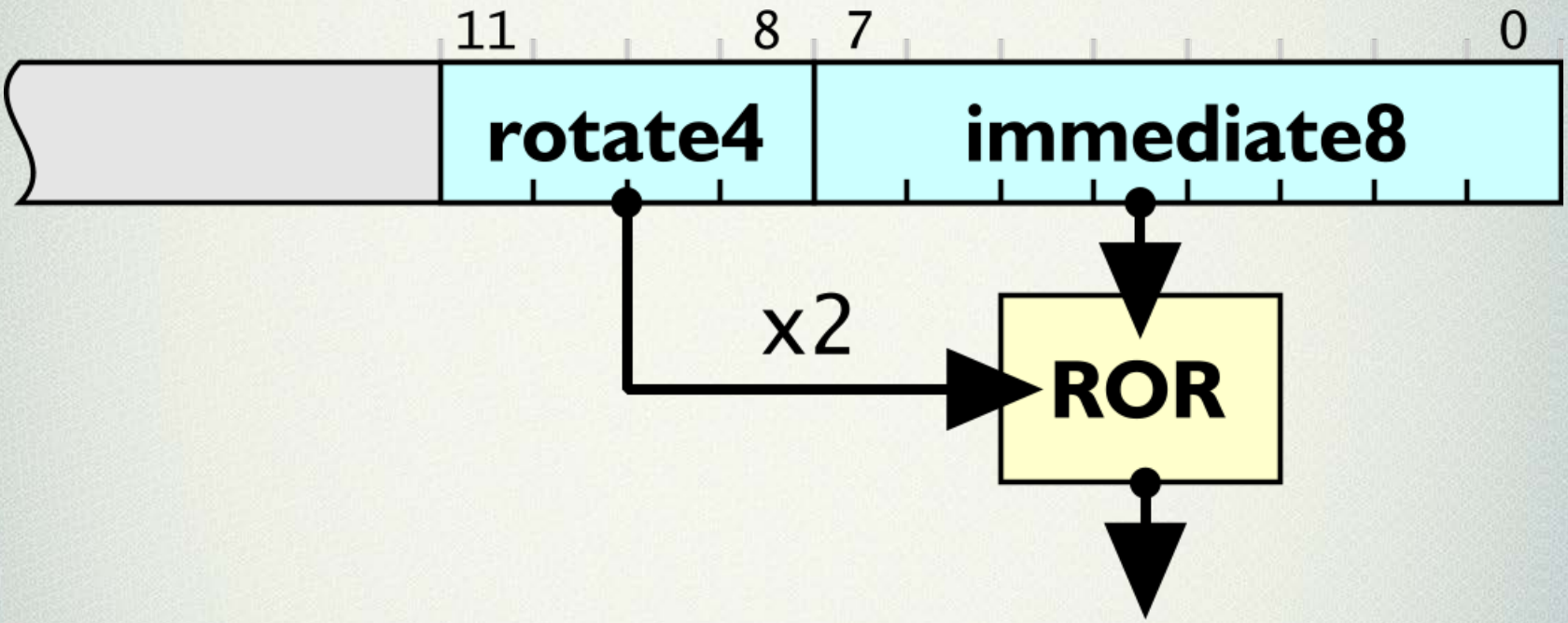
MOV Instruction Example

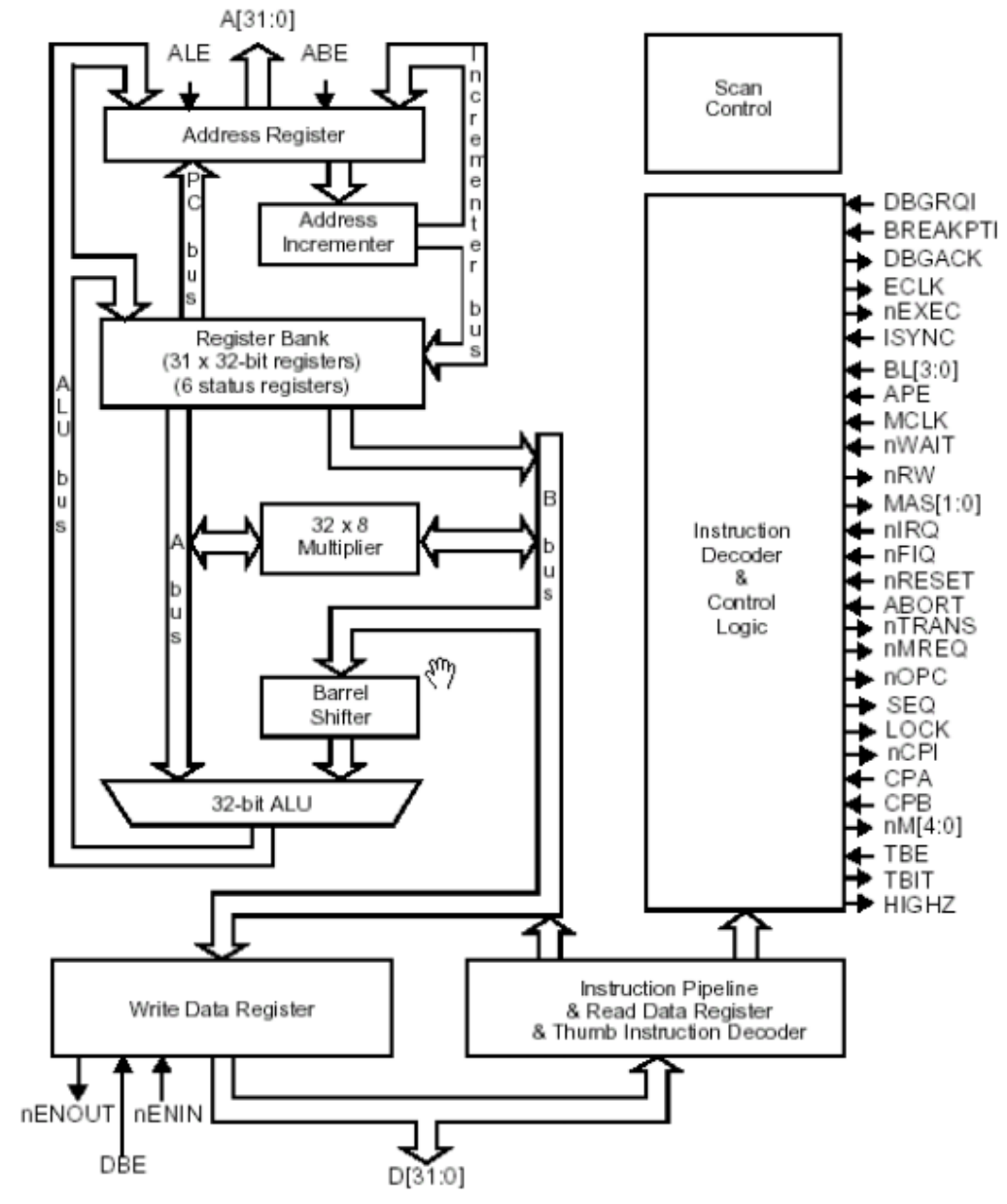
- Bit pattern : 0xE3A004FF
- Mnemonic : MOV r0, #0xFF, 8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction Type
1	1	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	MOV r0, #0xFF,8
Condition							OPCODE				S	SBZ				Rd =r0				Rotate_imm				8_bit_immediate								

- Condition =E means ALWAYS (executed)
- The value in bits [11:8] of the instruction specify the rotate right value which is multiplied by 2. Hence you cannot rotate right by an odd number.

Rotate Right Shifter




A[31:0]
ALE
ABE
Address Register
Incrementer bus
PC bus
Address Incrementer
Register Bank
(31 x 32-bit registers)
(6 status registers)
ALU bus
A bus
B bus
32 x 8
Multiplier
Barrel
Shifter
32-bit ALU
Scan
Control
Instruction
Decoder
&
Control
Logic
DBGRQI
BREAKPTI
DBGACK
ECLK
nEXEC
ISYNC
BL[3:0]
APE
MCLK
nWAIT
nRW
MAS[1:0]
nIRQ
nFIQ
nRESET
ABORT
nTRANS
nMREQ
nOPC
SEQ
LOCK
nCPI
CPA
CPB
nM[4:0]
TBE
TBIT
HIGHZ
Write Data Register
Instruction Pipeline
& Read Data Register
& Thumb Instruction Decoder
nENOUT
nENIN
DBE
D[31:0]

MOV Instruction

- As only 12 bits are available, they cannot represent all possible 32-bit numbers. Hence some numbers cannot be represented.
- What are the numbers that cannot be represented?????
- Can 0x55555555 be represented?
- How about 0xFFFFFFFF?

More MOV Examples

- MOV r0, #0xFF ; r0 =255, no rotation ;needed
- MOV r0, #0x1, 30 ; r0 = 4, rotation of 30 bits ;to the right is same as rotating 2 bits to the ;left or multiplying by 4.
- MOV r0, #0x1, 26 ; r0 = 64, rotation of 26 bits ;to the right is same as rotating left by 6 bits ;or multiplying by 64

Example

- Calculate the rotation necessary to generate the constant 4080 using the byte rotation scheme.
- $4080 = 111111110000_2$
- The byte 11111111_2 or 0xFF can be rotated by 4 bits to the left.
- Rotation of 4 bits to the left is equivalent to rotating (32-4) bits or 28 bits to the right
- MOV r0, #0xFF, 28 ; r0 = 4080

Constants Created By Rotation

- Examples
  - **0 - 255** **[0 - 0xff]**
  - **256,260,264,..,1020** **[0x100-0x3fc, step 4, 0x40-0xff ror 30]**
  - **1024,1040,1056,..,4080** **[0x400-0xff0, step 16, 0x40-0xff ror 28]**
  - **4096,4160, 4224,..,16320** **[0x1000-0x3fc0, step 64, 0x40-0xff ror 26]**

- These can be loaded using, for example:
  - **MOV r0, #0x40, 26** **; => MOV r0, #0x1000** **(ie 4096)**

- To make this easier, the assembler will convert to this form for us if simply given the required constant:
  - **MOV r0, #4096** **; => MOV r0, #0x1000** **(ie 0x40 ror 26)**

MVN

- MVN (move negative) instruction moves a one's complement of the operand into a register, can also be used to generate classes of numbers, such as
- MVN r0, #0 ; r0 = 0xFFFFFFFF
- MVN r0, #0xFF, 8 ; r0 = 0x00FFFFFF

Keil μvision

- Keil is a very intelligent assembler, instead of writing "MOV r0, #0xFF, 28" to move 4080 into register r0, you can simply write "MOV r0, #4080" and let the assembler figure out how to assemble it correctly.
- The Keil assembler will first try to use a MOV instruction with the correct shift to represent the constant, if it is not successful, it will try to use MVN to represent the constant. Only if both methods fail, then an assembly error is produced.

LDR

- The best way to load a constant into a register is to use a pseudo-instruction LDR. It is easiest and create the least headache for the programmer.
- LDR <Rd>, <numeric constant>
- This method can load all possible 32-bit constants.

Examples

- LDR r0, =42; Keil assembler will translate this ;into => MOV r0, #42
- LDR r2, =0xFFFFFFFF; assembled into

  ;MVN r2, #0
- LDR r1 = 0x55555555; this number cannot be ;represented using a MOV or MVN instruction ;and the constant has to be stored in a Literal ;pool in the memory. This get assembled into => ;LDR r1, [PC, #N] where N is the offset to the ;literal pool.

Literal Pool

- The literal pool is normally located at the END directive. However if the location is more than 4KB away (unlikely for this course since our programs are very small) from the place where it is used, and error will occur.
- To prevent the error, the literal pool can be located nearer using the LTORG assembler directive.

Pseudo-instruction ADR

- ADR is another useful pseudo-instruction for loading the 32-bit address into a register.
- The 32-bit address is stored in the memory which can then be retrieved by accessing through a relative offset from the PC register value of the instruction.
- For addresses further away, ADRL can be used
- Example:

```
ADR r1,label1        ; range < 255Bytes
ADRL r2,labe12       ; range < 64K
```

Addresses that are far away

- What to do if the addresses are more than 64Kbytes?
- Again we have to use the LDR instuction
- Format : LDR <Rd>, =label
- Example : LDR r0, =data_array

  where data_array is a label.
- This method is different from ADR and ADRL in that labels outside of a section can be referenced and the linker will resolve the reference during linking time.

Loading Address Summary

- Use ADR <Rd>, label whenever possible, The address cannot be more than 255 bytes/words away if the address is not word aligned/word aligned.
- If the above cannot be used, use the ADRL pseudo-instruction, which will calculate an offset that is within ±64KB/256KB range for an address that is not word aligned/word aligned. It is translated into two operations.
- Use pseudo-instruction LDR <Rd>, =label for referencing labels in other sections of code. Slowest as extra cycles are used to fetch the literal from memory.

Summary

- Understand how the ARM's MOV instruction works.
- Understand how MVN works.
- Pseudo-instruction LDR for loading constants
- Pseudo-instruction ADR for loading addresses