Week 9

Conditional Branch Instruction → tests flag bits in CPSR; takes the branch if the tested condition is true

- 8 possible conditional branches based on the state of a single bit
    - 4 branch on true, 4 branch on false

Encoding ARM Assembly Language = converting from ARM assembly to machine language

Encoding Branch (4 bytes):

- 4 bits = condition (equal, not equal, greater than, etc.)
- 3 bits = show that this instruction is a branch instruction (101)
- 1 bit = L bit (if L = 0, normal branch; if L = 1, branch with link instruction)
- 24 bits = signed program-counter relative offset (word)
    - To find word offset, you could the number of instructions from the branch instruction to the "else" instruction
    - You then input the negative two's complement of that number as the signed word offset in 24 bits

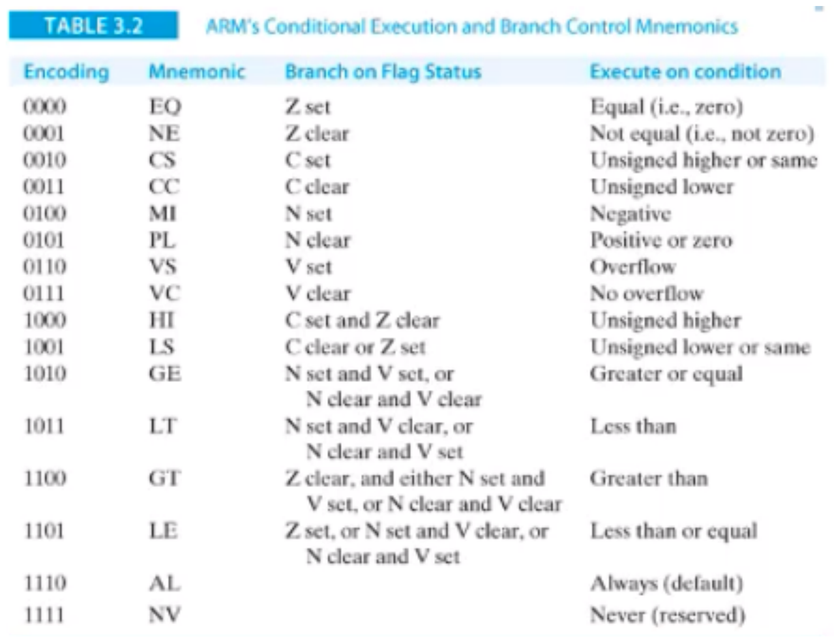


TABLE 3.2 ARM's Conditional Execution and Branch Control Mnemonics

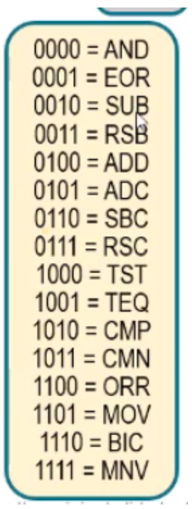
Encoding	Mnemonic	Branch on Flag Status	Execute on condition
0000	EQ	Z set	Equal (i.e., zero)
0001	NE	Z clear	Not equal (i.e., not zero)
0010	CS	C set	Unsigned higher or same
0011	CC	C clear	Unsigned lower
0100	MI	N set	Negative
0101	PL	N clear	Positive or zero
0110	VS	V set	Overflow
0111	VC	V clear	No overflow
1000	HI	C set and Z clear	Unsigned higher
1001	LS	C clear or Z set	Unsigned lower or same
1010	GE	N set and V set, or N clear and V clear	Greater or equal
1011	LT	N set and V clear, or N clear and V set	Less than
1100	GT	Z clear, and either N set and V set, or N clear and V clear	Greater than
1101	LE	Z set, or N set and V clear, or N clear and V set	Less than or equal
1110	AL		Always (default)
1111	NV		Never (reserved)

Note: convert output to hexadecimal

Data-Processing (4 bytes)

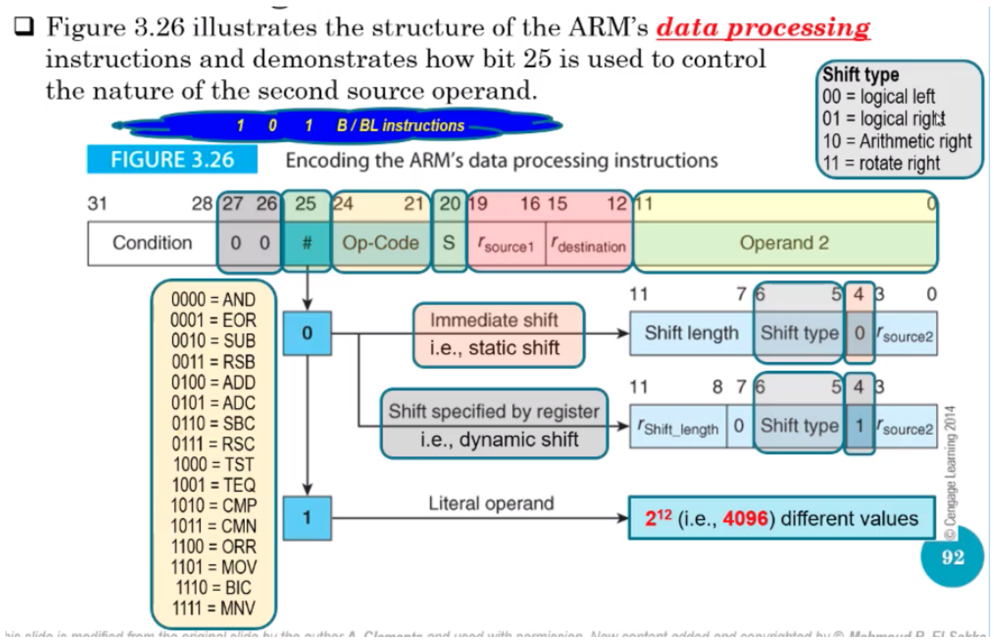
- 4 bits =
- 2 bits = 00 (show that this instruction is data processing)
- 1 bit = bit 25

    - if bit 25 = 1, the operand2 is a literal
    - If bit 25 = 0, operand2 is a static or dynamic shift
- 4 bits = Op-Code



0000	AND
0001	EOR
0010	SUB
0011	RSB
0100	ADD
0101	ADC
0110	SBC
0111	RSC
1000	TST
1001	TEQ
1010	CMP
1011	CMN
1100	ORR
1101	MOV
1110	BIC
1111	MNV

- 1 bit = S (whether instruction has an S on the end or not)
- 4 bits = rsource
- 4 bits = rdestination
- 12 bits = Operand 2 (can be literal, register with shift, etc.)
    - Based on bit 25



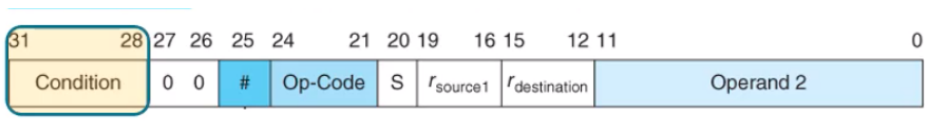
❑ Figure 3.26 illustrates the structure of the ARM's ***data processing*** instructions and demonstrates how bit 25 is used to control the nature of the second source operand.

FIGURE 3.26 Encoding the ARM's data processing instructions

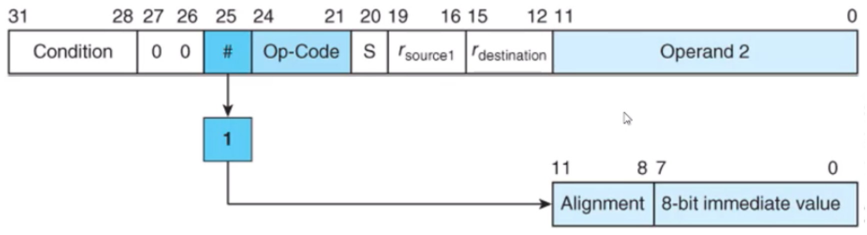
Note: if you're using CMP, TEST, TEQ, and CMN S = 1 and set rdestination to 0000
Note: with MOV instructions, rsource1 = 0000
Note: when shift type is rotate right and shift length is 0, this is RRX

Conditional Execution:

- 4 bits = condition (equal, not equal, etc.)
- Bit 27, 26 = 0
- Bit 25 = 1 or 0
- Bit 24-21 = Op-code (for data processing instruction)



Encoding Literals:
We can represent 4096 different values for literals because operand2 is a 12-bit field.



Note: when you convert your hexadecimal literal to binary, count the number of bits between the 2 ones (include the ones); if the distance is less than or equal to 8 or more than or equal to 24, you will be able to encode this number

Decoding ARM Assembly Language = converting from machine language to ARM assembly

Addressing Modes

- Literal
  - Using a literal/constant
  - ADD r0,r1,#Q OR [r0] ← [r1] + Q
- Direct (absolute)
  - Loading the content of memory-location Mem into register
  - Eg. LDR r0,Mem OR [r0] ← [Mem]
  - Note: this mode not supported by ARM
- Register Indirect
  - Load register with content of memory location pointed at by r1
  - Also known as indexed/pointer-based
  - Eg. LDR r0,[r1] OR [r0] ← [[r]]
  - Indirect addressing uses square brackets ([])
  - ARM supports memory-addressing where address of an operand is computed by adding the contents of a register to a literal offset encoded into the instruction
    - Called **base plus displacement addressing**
    - Eg. LDR r0, [r1, #4] (sum of content of pointer (the address) plus 4)

        - Literal offset is a true 12-bit literal ranging from 0-4095
- Program Counter Relative Addressing
    - A special type of indirect addressing where r15 (the program counter) is used as a pointer
    - Location of data is relative to location of instruction; constant
    - R15 (PC) is incremented after you perform an instruction on it because of the nature of the program counter
    - Can be static or dynamic
        - Static means offset is given as a literal
        - Dynamic means offset is given in terms of another register; modified at runtime
            - [r2] ← [[r0] + [r1]]
            - Known as **register indirect addressing with base and index registers**
            - Can apply shifts to registers that are offsets
                - LDR r2, [r0,r1,LSL #2]
                - Known as **register indirect addressing with base and index registers + scaling**
    - Auto-indexing Addressing Mode
        - Allows us to access element and update its index at the same time (can be done before or after accessing element)
        - Pointer is automatically adjusted to point at the next element before or after it is used
        - In ARM, we can increment or decrement by any value, not just 1
        - **Auto-indexing Pre-indexing Addressing Mode**
            - Increments base register by an offset, accesses operand at location by the updated base register
            - Indicated by adding a '!' to the end of the address
                - Eg. LDR r0,[r1,#8] !
        - **Auto-indexing Post-indexed addressing mode**
            - First access operand at location, then increment base register
            - Denoted by placing the offset outside the square
                - Eg. LDR r0, [r1], #8