Amrita School Of Engineering, Bangalore Campus

ARM Assembly Language Programming

PREVIOUSLY

- Programmer's Model
- Instruction Set

TODAY…

- Data processing instruction

DATA PROCESSING INSTRUCTION

- Basically operations are arithmetic, logical or movement of data.
- All operands are 32 bit wide
- If any result is obtained that is 32 bit wide stored in destination register.
- 3 address instruction format.

ADDITION

- ADD r0,r1,r2 ;r0 = r1 + r2
- ADC r0,r1,r2 ;r0 = r1 + r2 + C(where is C??)
- SUB r0,r1,r2 ;r0 = r1 - r2
- SBC r0,r1,r2 ;r0 = r1 - r2 + C - 1 (B=~C)
- RSB r0,r1,r2 ; r0 = r2 – r1;
- RSC r0,r1,r2 ;r0 = r2 – r1 + C -1

BIT WISE LOGICAL OPERATION

- AND r0,r1,r2 ;r0 = r1 & r2
- ORR r0,r1,r2 ;r0 = r1 | r2
- EOR r0,r1,r2 ;r0 = r1 ^ r2
- BIC r0,r1,r2 ;r0 = r1 &(~r2)

- BIC – Bit Clear

REGISTER MOVEMENT OPERATIONS

- MOV r0,r1 ;r0 ← r1
- MVN r0,r1 ;r0 ← (~r1)
  - Move negated

COMPARISON OPERATION

- These instruction does not produce result instead they set conditional codes(N, Z, C, V)
- CMP r1,r2 ;set cc on r1 – r2
- CMN r1,r2 ;set cc on r1 + r2
- TST r1,r2 ;set cc on r1 & r2 (bit test)
- TEQ r1,r2 ;set cc on r1 ^ r2 (test equal)

IMMEDIATE OPERANDS

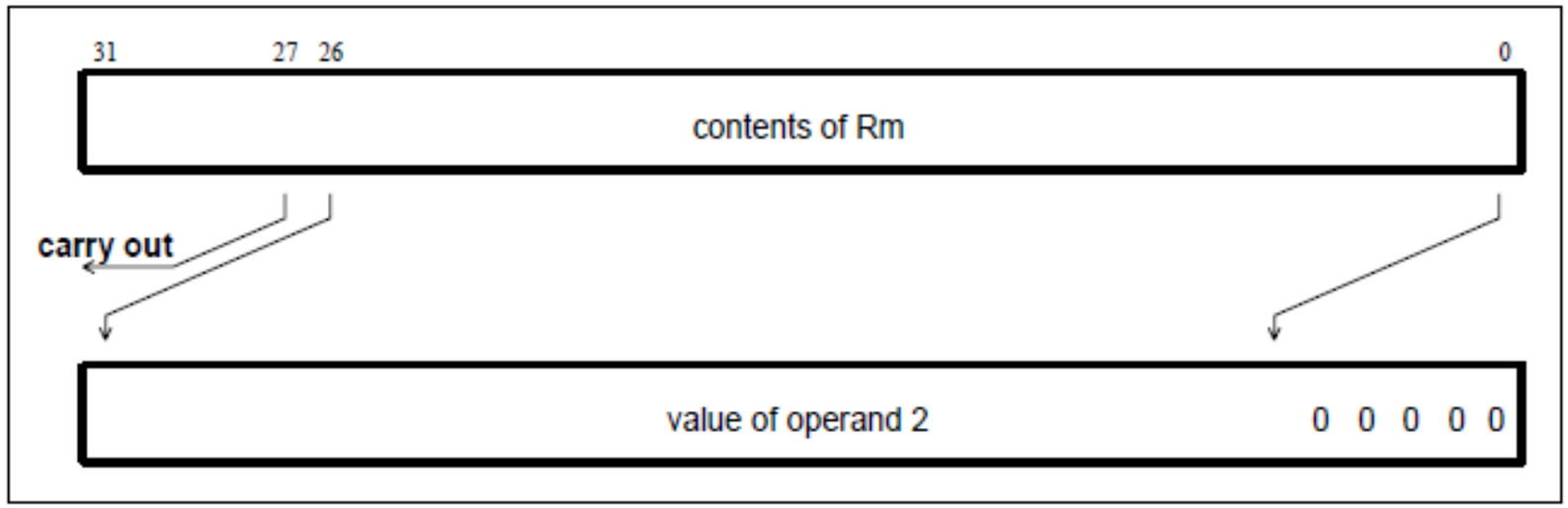
- ADD r3,r3,#1
- AND r8,r7,# &FF
- # notation used for immediate field
- & notation is used for representing hexadecimal numbers
- How many bit immediate possible ??
  - Covered later…

SHIFTED REGISTER OPERANDS

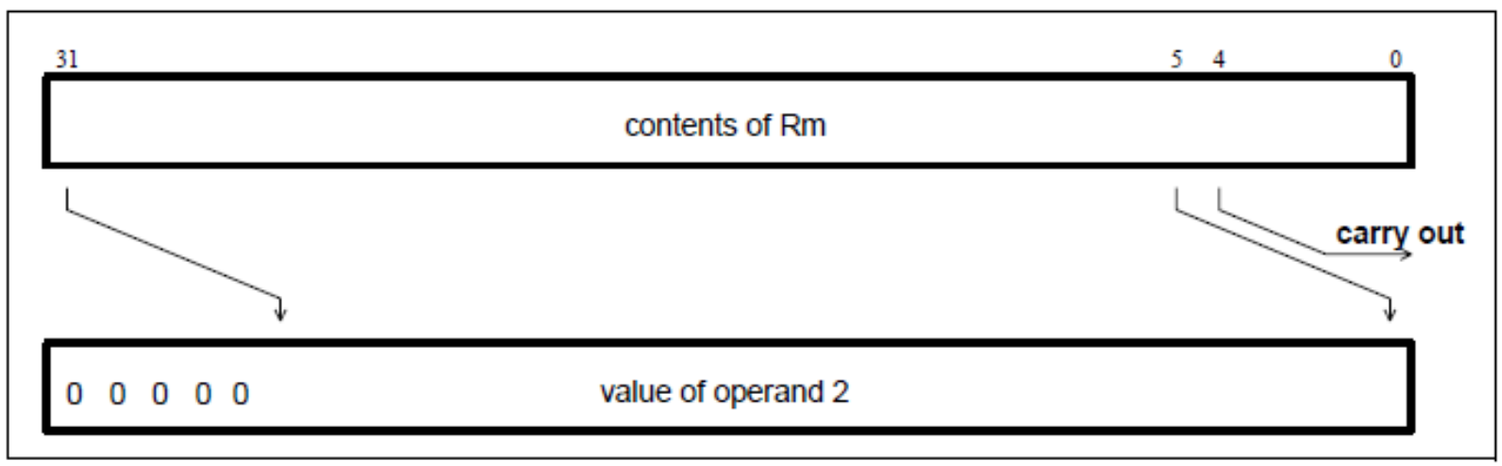
- ADD r3,r2,r1,LSL #3 ;r3 = r2 + (8xr1)
- Various shifting operations are
  - LSL – logical shift left
  - LSR – logical shift right
  - ASL – Arithmetic shift left = LSL
  - ASR – Arithmetic shift right
  - ROR – Rotate right
  - RRX – Rotate right extended by 1 place
- Register values can be used to specify the number of bits to be shifted
  - ADD r5,r5,r3, LSL r2 ; r5 = r5+r3x(2^r2)

SHIFTING

- Logical Shift Left

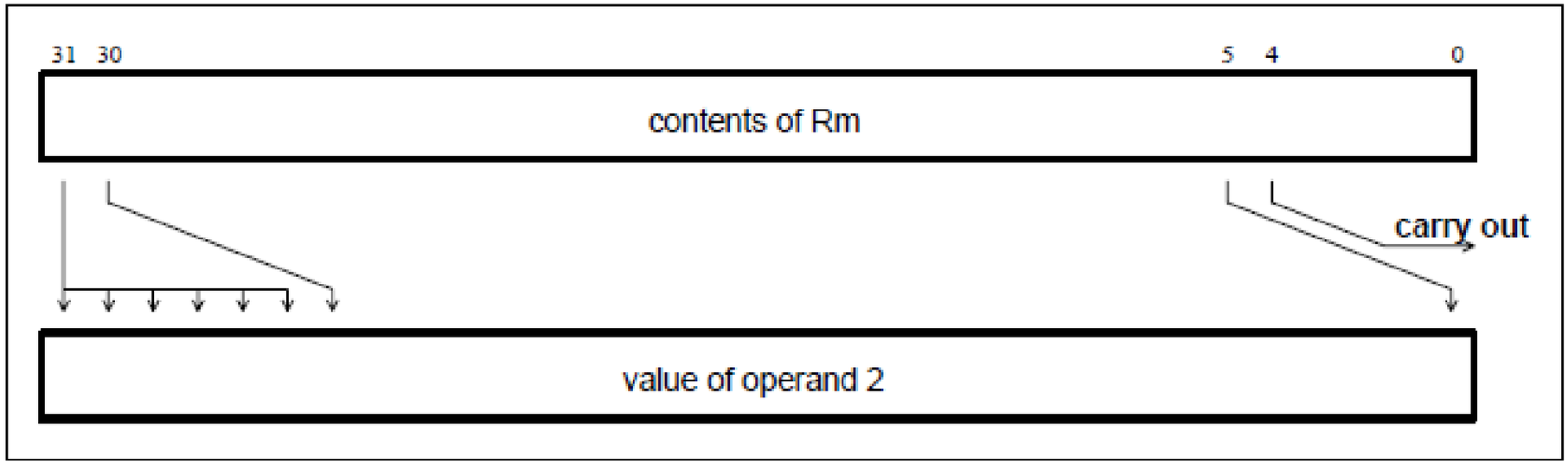


- Logical Shift Right



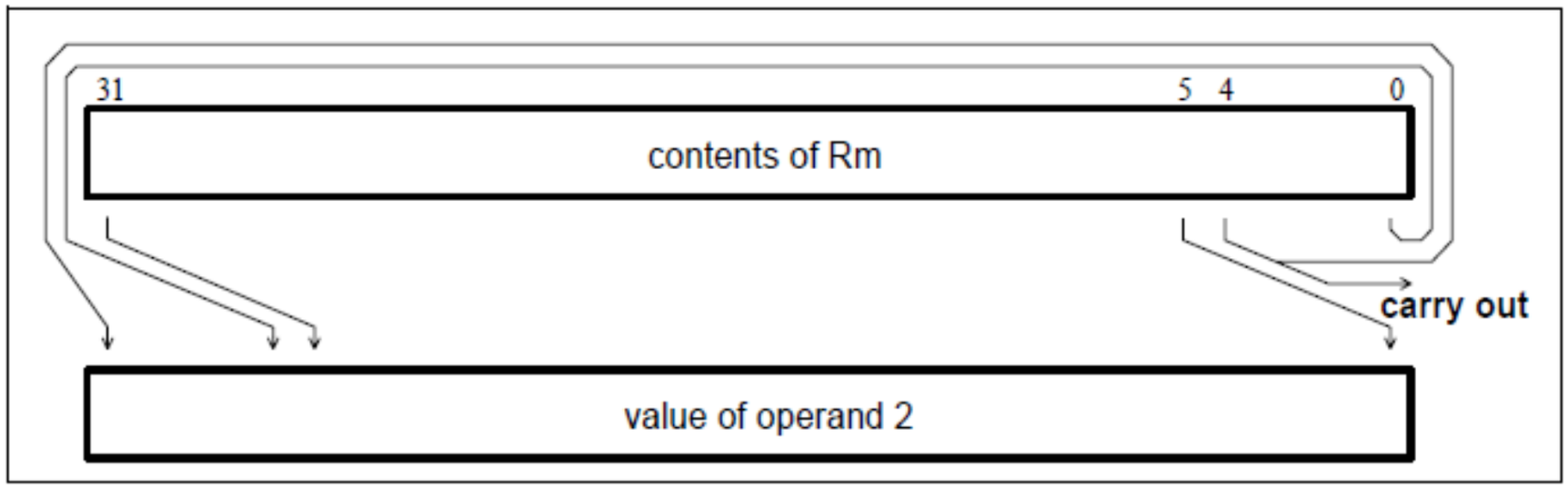
CONTD...

- Arithmetic Shift Right

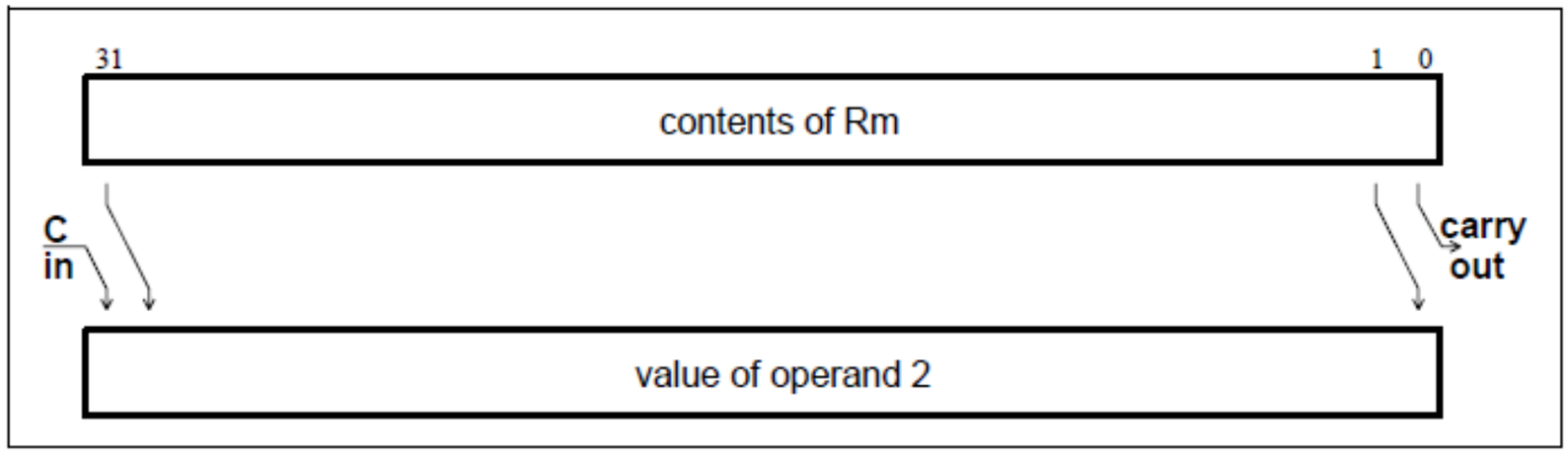


CONTD...

- Rotate Right



- Rotate Right Extended



SETTING THE CONDITION CODES

- Any data processing instruction can set the condition codes if programmer wishes to
- Just add ‘S’ in the instruction
- S stands for set condition code
- ADDS r2,r2,r0
- Arithmetic instruction set – N,Z,C,V
- Logical and move instruction set – N,Z

CONDITION EXECUTION

- An unusual feature for ARM instruction is that every instruction can be executed conditionally
- Similar to conditional branches ARM extends this to all its instruction
- To execute an instruction conditionally, simply postfix it with the appropriate condition

CONTD...

- ARM instructions can be made to execute conditionally by postfixing them with the appropriate condition code field.
  - This improves code density *and* performance by reducing the number of forward branch instructions.

```
    CMP   r3,#0
    BEQ   skip
    ADD   r0,r1,r2
skip
```

```
    CMP   r3,#0
ADDNE r0,r1,r2
```

- By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using "S". CMP does not need "S".

```
loop
  …
  SUBS r1,r1,#1
  BNE loop
```

decrement r1 and set flags

if Z flag clear then branch

COND FIELD

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Zset
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	Cset
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	Ml	Minus / negative	Nset
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	Vset
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE-	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

COND FIELD

- NV: The 'never' condition (NV) should not be used - there are plenty of other ways to write no-ops (instructions that have no effect on the processor state) in ARM code.

- The reason to avoid the 'never' condition is that (read the book page no. 125 just before table 5.3)

COND FIELD

- Alternative mnemonics: There is more than one way to interpret the condition field.
- CS or HS:Both cause the instruction to be executed only if the C bit in the CPSR is set.
  - The alternatives are available because the same test is used in different circumstances. If you have previously added two unsigned integers and want to test whether there was a carry-out from the addition, you should use CS.
  - If you have compared two unsigned integers and want to test whether the first was higher or the same as the second, use HS.
- The alternative mnemonic removes the need for the programmer to remember that an unsigned comparison sets the carry on higher or the same.

EXAMPLES OF CONDITIONAL EXECUTION

- Use a sequence of several conditional instructions

  if (a==0) func(1);

- Set the flags, then use various condition codes

  if (a==0) x=0;

  if (a>0) x=1;

- Use conditional compare instructions

  if (a==4 || a==10) x=0;

```
CMP r0,#0
MOVEQ r0,#1
BLEQ func

CMP r0,#0
MOVEQ r1,#0
MOVGT r1,#1

CMP r0,#4
CMPNE r0,#10
MOVEQ r1,#0
```

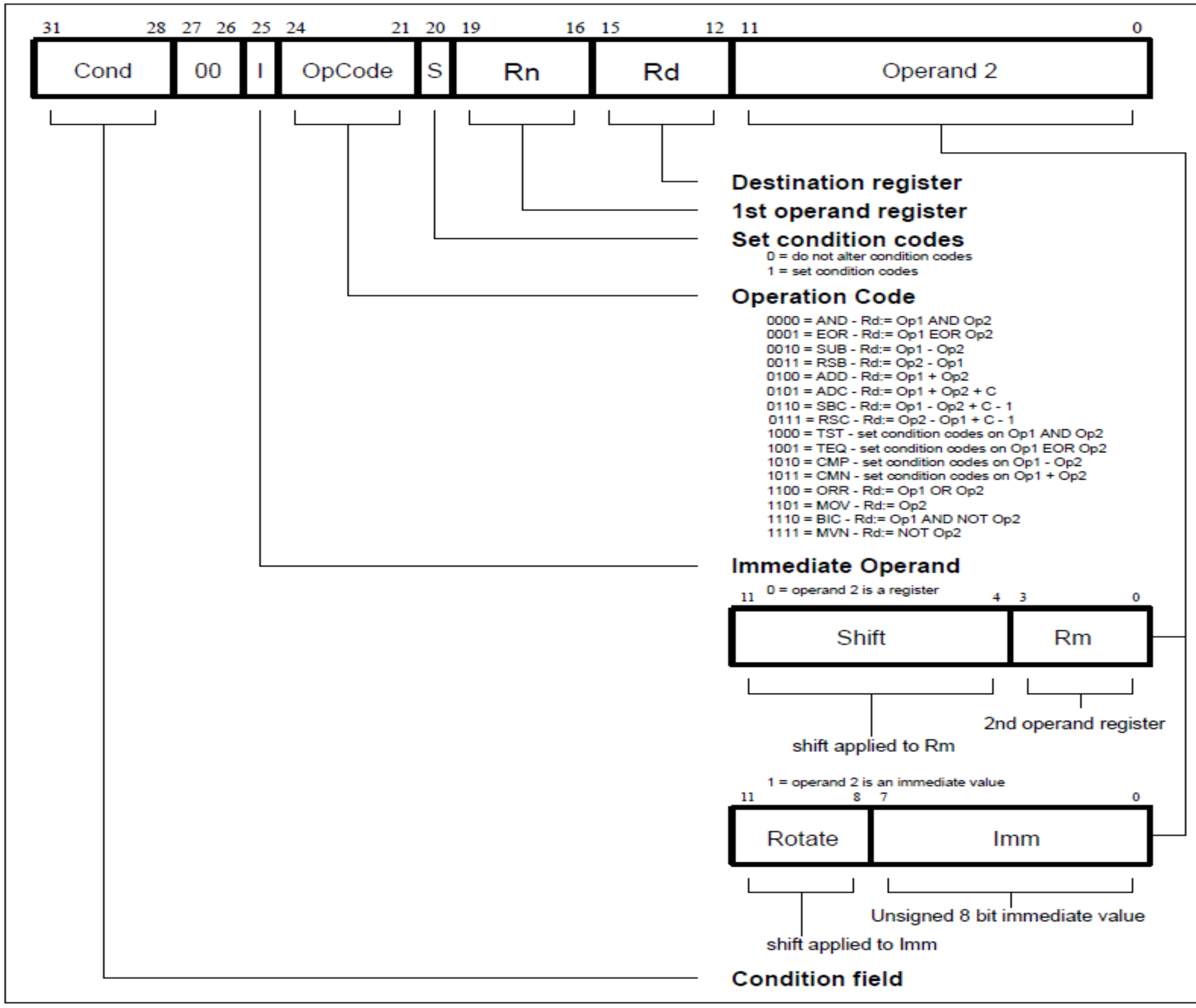
EXAMPLE: CONDITIONAL INSTRUCTION (Later)

- true block
- MOVLT r0,#5 ; generate value for x
- ADRLT r4,x ; get address for x
- STRLT r0,[r4] ; store x
- ADRLT r4,c ; get address for c
- LDRLT r0,[r4] ; get value of c
- ADRLT r4,d ; get address for d
- LDRLT r1,[r4] ; get value of d
- ADDLT r0,r0,r1 ; compute y
- ADRLT r4,y ; get address for y
- STRLT r0,[r4] ; store y

We will revisit this slide.

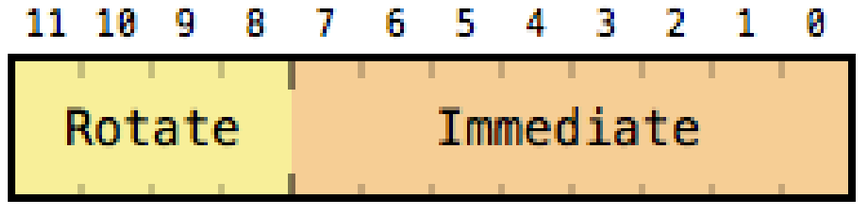
INSTRUCTION ENCODING

6th May 2020

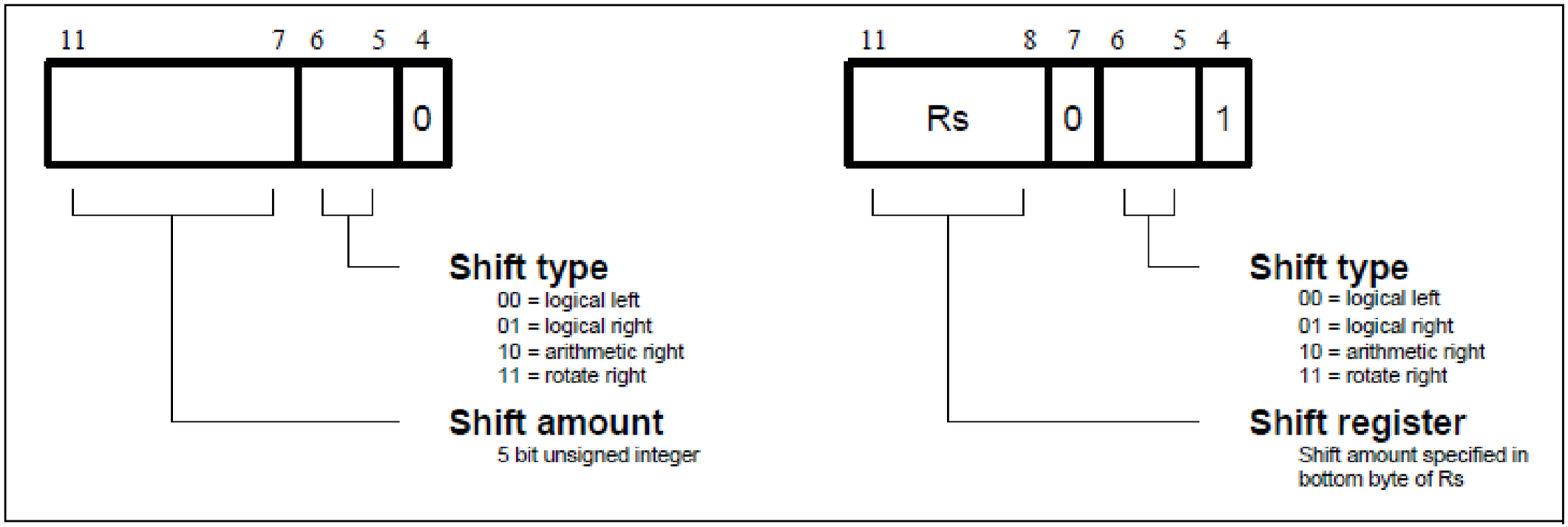

31
28
27
26
25
24
21
20
19
16
15
12
11
0
Cond
00
I
OpCode
S
Rn
Rd
Operand 2
Destination register
1st operand register
Set condition codes
0 = do not alter condition codes
1 = set condition codes
Operation Code
0000 = AND - Rd:= Op1 AND Op2
0001 = EOR - Rd:= Op1 EOR Op2
0010 = SUB - Rd:= Op1 - Op2
0011 = RSB - Rd:= Op2 - Op1
0100 = ADD - Rd:= Op1 + Op2
0101 = ADC - Rd:= Op1 + Op2 + C
0110 = SBC - Rd:= Op1 - Op2 + C - 1
0111 = RSC - Rd:= Op2 - Op1 + C - 1
1000 = TST - set condition codes on Op1 AND Op2
1001 = TEQ - set condition codes on Op1 EOR Op2
1010 = CMP - set condition codes on Op1 - Op2
1011 = CMN - set condition codes on Op1 + Op2
1100 = ORR - Rd:= Op1 OR Op2
1101 = MOV - Rd:= Op2
1110 = BIC - Rd:= Op1 AND NOT Op2
1111 = MVN - Rd:= NOT Op2
Immediate Operand
0 = operand 2 is a register
11
4
3
0
Shift
Rm
2nd operand register
shift applied to Rm
1 = operand 2 is an immediate value
11
8
7
0
Rotate
Imm
Unsigned 8 bit immediate value
shift applied to Imm
Condition field

ARM IMMEDIATE VALUE ENCODING

- The ARM instruction set encodes immediate values in an unusual way.
- Despite only using 12 bits of instruction space, the immediate value can represent a useful set of 32-bit constants.
- ARM doesn't use the 12-bit immediate value as a 12-bit number. Instead, it's an 8-bit number with a 4-bit rotation, like this:



SHIFTING (When second operand is register)



SHIFT AMOUNT

- Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.
- If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.
- If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

SPECIAL CASE

- LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand.
- LSR #0 & ASR #0 is Equated to LSL #0 by assembler
- If shift value is greater than or equal to 32 ??

CONTD...

- LSL by 32 has result zero, carry out equal to bit 0 of Rm.
- LSL by more than 32 has result zero, carry out zero.
- LSR by 32 has result zero, carry out equal to bit 31 of Rm.
- LSR by more than 32 has result zero, carry out zero.
- ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.

CONTD…

- ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.
- ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

(take a small example and work it out as HW)

MULTIPLY

- Difference from other Arithmetic instruction
  - Immediate second operand are not supported
  - Result register must not be same as first source register
  - Multiply give 64bit result, least 32 bit is stored in result register rest is ignored
- However multiply long instruction can store all 64 bit result.

CONTD…

- The Basic ARM provides two multiplication instructions.
- Multiply
  - MUL{<cond>}{S} Rd, Rm, Rs ; Rd = Rm * Rs
- Multiply Accumulate - does addition for free
  - MLA{<cond>}{S} Rd, Rm, Rs,Rn ; Rd = (Rm * Rs) + Rn

MULTIPLY-LONG AND MULTIPLY-ACCUMULATE LONG

- Instructions are
  - MULL which gives RdHi,RdLo:=Rm*Rs
  - MLAL which gives RdHi,RdLo:=(Rm*Rs)+RdHi,RdLo
- However the full 64 bit of the result now matter (lower precision multiply instructions simply throws top 32bits away)
  - Need to specify whether operands are signed or unsigned
- Therefore syntax of new instructions are:
  - UMULL{<cond>}{S} RdLo,RdHi,Rm,Rs
  - UMLAL{<cond>}{S} RdLo,RdHi,Rm,Rs
  - SMULL{<cond>}{S} RdLo, RdHi, Rm, Rs
  - SMLAL{<cond>}{S} RdLo, RdHi, Rm, Rs

EXAMPLE

- Example (Multiply, Multiply-Accumulate)

MUL r4, r3, r2	r4 := $[r3 \times r2]_{<31:0>}$
MLA r4, r3, r2, r1	r4 := $[r3 \times r2 + r1]_{<31:0>}$

- Note
  - least significant 32-bits are placed in the result register, the rest are ignored
  - immediate second operand is not supported
  - result register must not be the same as the first source register
  - if `S` bit is set the V is preserved and the C is rendered meaningless
- Example (r0 = r0 x 35)
  -

EXAMPLE (quick Quiz)

Can you do r0 = 5 r0

Can you do r0 = 7 r0

ADD r0, r0, r0, LSL #2 :=5xr0

RSB r0, r0, r0, LSL #3 : = 7xr0

SUMMARY

- MUL r4,r3,r2 ; r4 = r3 x r2 [31:0]
- MLA r4,r3,r2,r1; r4 = r3 x r2 +r1 [31:0]
- UMULL; rdHi:rdLo = Rm x Rs
- UMLAL; rdHi:rdLo += Rm x Rs
- SMULL; rdHi:rdLo = Rm x Rs
- SMLAL; rdHi:rdLo += Rm x Rs

THANK YOU

6th May 2020