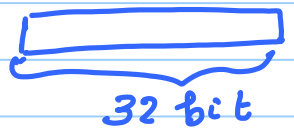
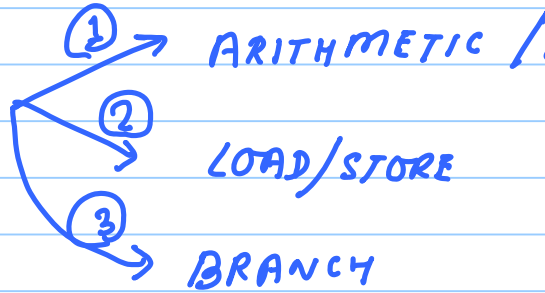
Aug - 17

Note Title 17-08-2012

Machine Representation

ARM Instruction → 



① ARITH / LOGICAL

[Short Cut - 1]

Extensions: if (j == 0)
a = b + c;

CPSR

① Short Cut

```
CMP  Rj, #0
BNE  .exit
ADD  Ra, Rb, Rc
```

suffix
ADD EQ

Do the addition iff the comparison before it detected an equality

② ADDEQ is a conditional instruction

```
CMP  Rj, #0
ADDEQ Ra, Rb, Rc
```

NE → Not equal
GT → Greater than
LT → Less than

GE → >=
LE → <=
PL → positive
MI → Negative

HI → unsigned higher
LO → unsigned lower.

? where does the cmp instruction store its result.

Ans: CPSR

Current Program Status Register

cmp (is actually doing a SUB)

- N → Negative
- Z → Zero
- C → Carry out
- F → Overflow

EQ	→	(Z == 1)
NE	→	(Z == 0)
GT	→	((N == 0) && (Z == 0))
GE	→	(N == 0)
LT	→	(N == 1)
LE	→	((N == 1) || (Z == 1))

cmp R_1, R_2 (R_1 = 5, R_2 = 3)

r = 5 - 3

5 - 3 = 2

5 == 3

GT

5 > 3

r ≥ 0

r ≠ 0

cmp R_1, R_2 (R_1 = 3, R_2 = 5)

r = 3 - 5

Short Cut -2

CMP is not the only instruction that can set the CPSR.

You can add a S suffix to any instruction to make it set the CPSR accordingly

C:

```
tmp = a+b;
if (tmp == 0)
    z = i+j;
```

4 inst. sequence

↓

2 inst. sequence

[ADDEQ / ADDS]

```
ADD   R_tmp, R_a, R_b
CMP   R_tmp, #0
ADDEQ R_z, R_i, R_j
```

↓ 2

```
ADDS  R_tmp, R_a, R_b
ADDEQ R_z, R_i, R_j
```

Shortcuts for Arith/Logi insts.

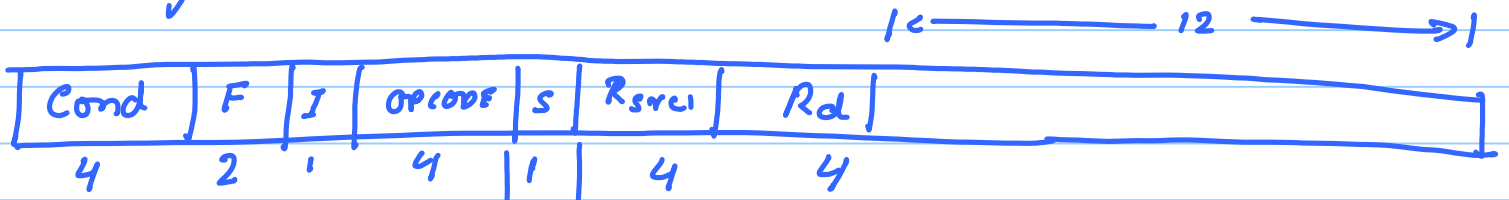
1. ADD R_1, R_2, R_3, LSL R_4

   $R_1 = R_2 + R_3 << R_4$

2. ADDEQ

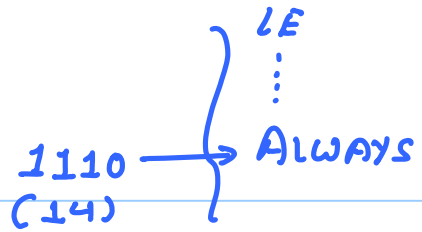
3. ADDS

Format of Assembly Insts.



- EQ
- NE
- GE
- LT

S bit



F : 00 → Arith / logical
01 → LD / STR
10 → Branch

I → is src2 an immediate (constant)

Opcode : 4 → 16 { MUL, ADD, SUB, : shift

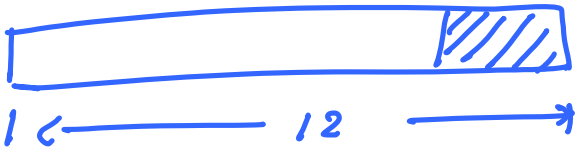
S : 1 : set cpsr
0 : Do not set cpsr

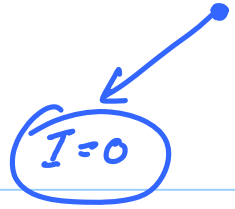
R_{src1} : Register for source 1

Rd : Register for destination

Remaining 12 bits.

Src2 - Use reg.

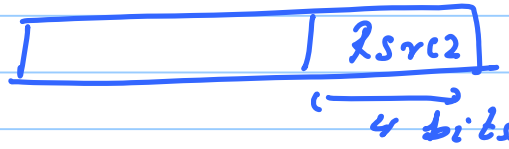




Simplest: ADD R_1, R_2, R_3

opcode — dest — src1 — src2

R_{src2} :

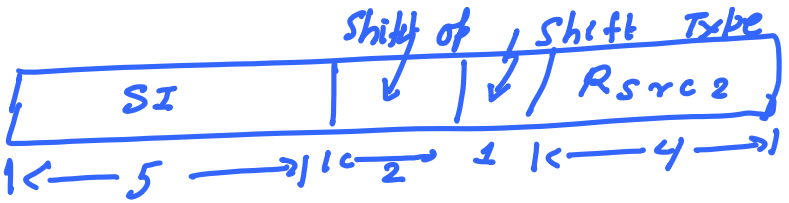


(A) ADD R_1, R_2, R_3, LSL R_4 ✓ ☺

⟷

(B) ADD ✓ R_1, R_2, R_3, LSL #2 ✓ ☺

We still have 8 bits left



decisions: differentiate between case (A) and (B)

(1)

Type of shift operation (2)

- LSL
- LSR
- ASR
- ROR

Shifting by an immediate:

use $\underbrace{SI}_{5} \rightarrow 0 \rightarrow 31$.

Shifting by a register: You need 4 bits

You have 5 bits ✓

(Problem Solved)

E.g.. ADD R_1, R_2, #10

(ADD R_1, R_2, : 20 bits; #10 : 12)

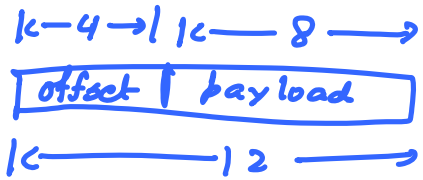
We have 12 bits for representing the immediate. But, an immediate can be very large (upto 32 bits)

We need to somehow compromise.

Aim:

Encode as many numbers as possible using the 12 bits that we have.

Arm Solution



8 bit part → payload

4 bit part → offset

[0-15] × 2 = [0, 2, 4 .. .. 28, 30]

Number = payload ROR (2 × offset)

TRICKY EXPECT EXAM Q

20

ADD R3, R2,

0x FE 00 00 00

payload

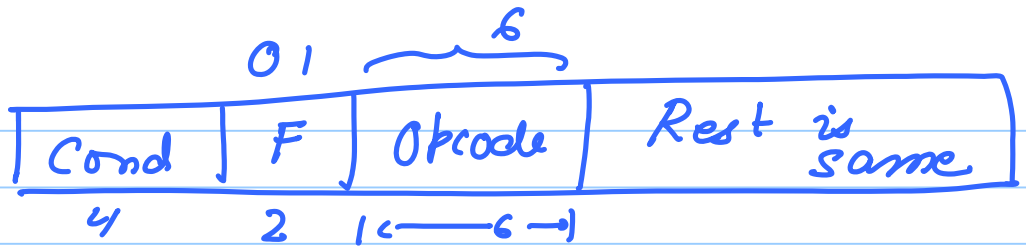
00 00 00 FE

0x E0 00 00 0F

0x 00 FE 00 00

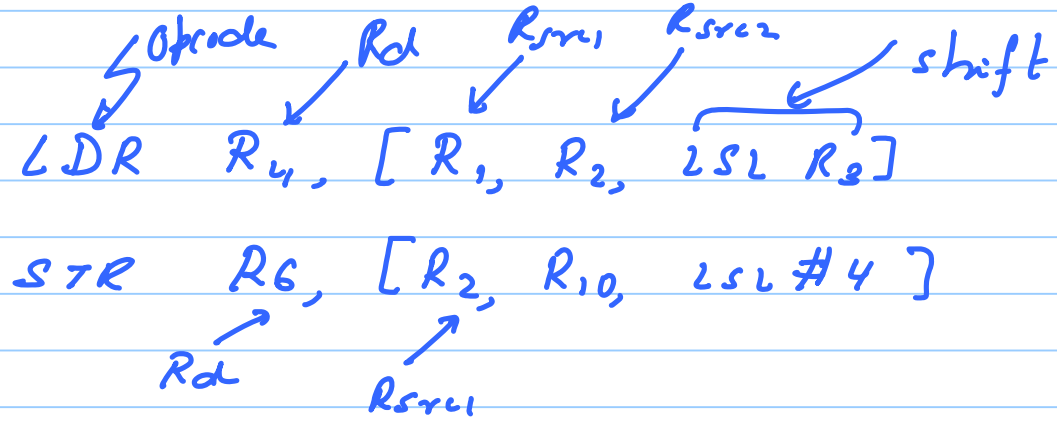
Read this in the book (Very Tricky)

LD/STR



I, S bits do not make sense for LDR/STR

Note



Pre-indexed

++i;

LDR R_1, [R_4, #4]!

$R_1 \leftarrow mem[R_4 + 4]$

$R_4 \leftarrow R_4 + 4$

Post-indexed

i++;

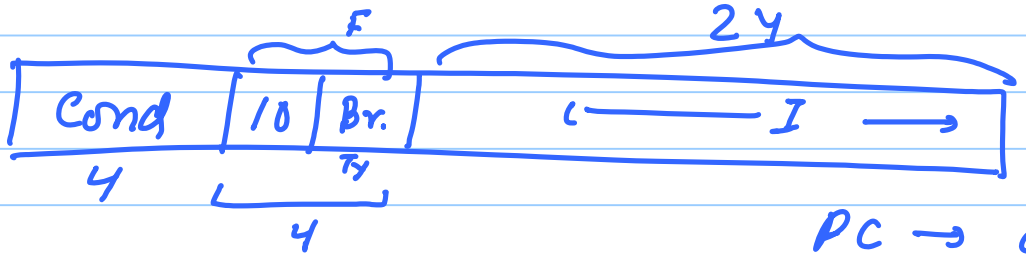
LDR R_1, [R_4], #4

$R_1 \leftarrow mem[R_4]$

$R_4 \leftarrow R_4 + 4$

Saving an increment and a shift in case of array accesses.

Branch.



Next Class

Adders -