SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Loading Constants, Conditional execution

Dr.G.Arthy
Assistant Professor
Department of EEE
SNS College of Engineering

Instruction Set

Instruction Type	Definition	Examples
MOVE	The contents of a register are copied to another.	MOVF, MOVWF, MOVLW
REGISTER	Register operations affect only a single register, and all except CLRW (clear W) operate on file registers.	CLRW, CLRF, DECF, INCF, SWAPF, COMF, RLF, RRF, BCF, BSF
ARITHMETIC	Addition and subtraction in binary gives the same result as in decimal or hex. .	ADDWF, ADDLW, SUBWF, SUBLW
LOGIC	Logic operations are carried out on bit pairs in two numbers to give the result which would be obtained if they were fed to the corresponding logic gate	ANDWF, ANDLW, IORWF, IORLW, XORWF, XORLW
TEST, SKIP & JUMP	make decisions (conditional program branches) which depend on some input condition or the result of a calculation	BTFSC, BTFSS, DECFSZ, INCFSZ, GOTO, CALL, RETURN, RETLW, RETFIE
CONTROL		NOP, SLEEP, CLRWDT

Loading constants

- There is no ARM instruction to move a 32-bit constant into a register.
- Since ARM instructions are 32 bits in size, they obviously cannot specify a general 32-bit constant.
- To aid programming there are two pseudo instructions to move a 32-bit value into a register.
- Here are the various loading constants
  - Load constant pseudo instruction LDR
  - Load address pseudo instruction ADR

Loading constants

LDR – Load constant pseudo instruction

- Syntax: LDR Rd, = constant

- Load constant pseudo instruction writes a 32-bit constant to a register using whatever instructions are available.
- It defaults to a memory read if the constant cannot be encoded using other instructions.

  - Rd <= 32 – bit constant

.

- Example: LDR r0, =0xff

Loading constants

Example: Loading the constant 0xff00ffff using an MVN..

- MVN r0, #0x00ff0000

Table 2: Before and after exécution of MVN instruction

PRE	POST
None	r0 = 0xff00ffff

Loading constants

ADR – Load address pseudo instruction

- Syntax: ADR Rd, label

- Load address pseudo instruction writes a relative address into a register, which will be encoded using a pc-relative expression.

- Rd <= 32 – bit relative address

- ADR instruction, or address relative instruction places the addres of the given label into register Rd, using a pc-relative add or subtra

CONDITIONAL EXECUTION

- Most ARM instructions are conditionally executed.
- The instruction only executes if the condition code flags pass a given condition or test.
- By using conditional execution instructions, the performance and code density can be increased.
- Conditional execution reduces the number of branches, which als reduces the number of pipeline flushes and thus improves the performance of the executed code.

CONDITIONAL EXECUTION

- Conditional execution depends upon two components:
  - Condition field
  - Condition flags

The condition field is located in the instruction, and

the condition flags are located in the CPSR.

- Example: ADD instruction with the EQ condition appended.
- This instruction will only be executed when the zero flag in the CPSR is set to 1.

ADDEQ r0, r1, r2

CONDITIONAL EXECUTION

```
CMP    R0, R1
BLT    .Lsmaller   @ if R0<R1 jump over
MOV    R2, R1      @ R1 is less than or equal to R0
B      .Lend       @ finish
.Lsmaller:
MOV    R2, R0      @ R0 is less than R1
.Lend:
```

```
CMP    R0, R1
MOVGE  R2, R1  @ R1 is less than or equal to R1
MOVLT  R2, R0  @ R0 is less than R1
```

CONDITIONAL EXECUTION

Add instruction	Condition
ADDEQ r3, r2, r1	Add if EQual
ADDNE r3, r2, r1	Add if Not Equal
ADDHS r3, r2, r1	Add if unsigned Higher or Same
ADDLO r3, r2, r1	Add if unsigned LOwer
ADDMI r3, r2, r1	Add if Minus (Negative)
ADDPL r3, r2, r1	Add if PLus (Positive or Zero)
ADDVS r3, r2, r1	Add if oVerflow Set
ADDVC r3, r2, r1	Add if oVerflow Clear
ADDHI r3, r2, r1	Add if unsigned HIgher
ADDLS r3, r2, r1	Add if unsigned Lower or Same
ADDGE r3, r2, r1	Add if signed Greater or Equal
ADDLT r3, r2, r1	Add if signed Less Than
ADDGT r3, r2, r1	Add if signed Greater Than
ADDLE r3, r2, r1	Add if signed Less than or Equal

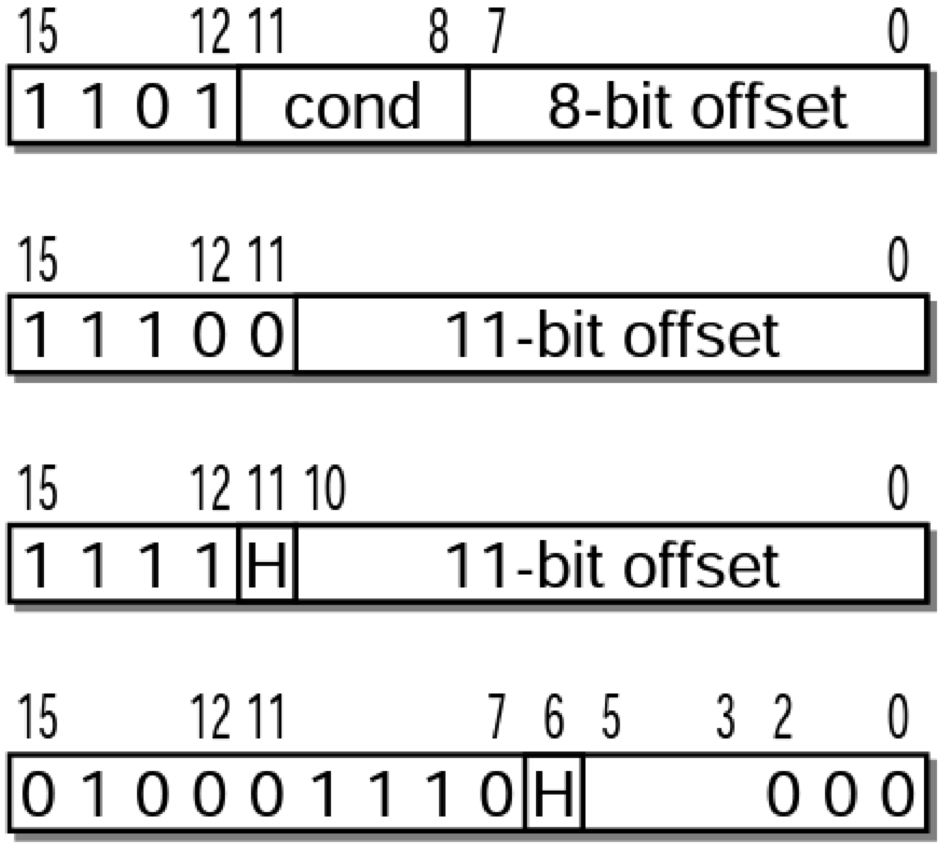
THUMB INSTRUCTIONS

- Thumb is:
  - a compressed, 16-bit representation of a subset of the ARM instruction set
    - primarily to increase code density
    - also increases performance in some cases
- It is not a complete architecture
  - all 'Thumb-aware' cores also support the ARM instruction set
    - therefore the Thumb architecture need only support common functions

THUMB BIT

- The 'T' bit in the CPSR controls the interpretation of the instruction stream
  - switch from ARM to Thumb (and back) by executing BX instruction
  - exceptions also cause switch to ARM code
    - return symmetrically to ARM or Thumb code
  - Note: do not change the T bit with MSR!

Thumb Branch Instructions



(1) B<cond> <label>

(2) B <label>

(3) BL <label>

(4) BX Rm

ARM AND THUMB INSTRUCTION SET

	ARM (*cpsr* $T = 0$)	Thumb (*cpsr* $T = 1$)
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution[a]	most	only branch instructions
Data processing instructions	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read-write in privileged mode	no direct access
Register usage	15 general-purpose registers $+pc$	8 general-purpose registers $+7$ high registers $+pc$

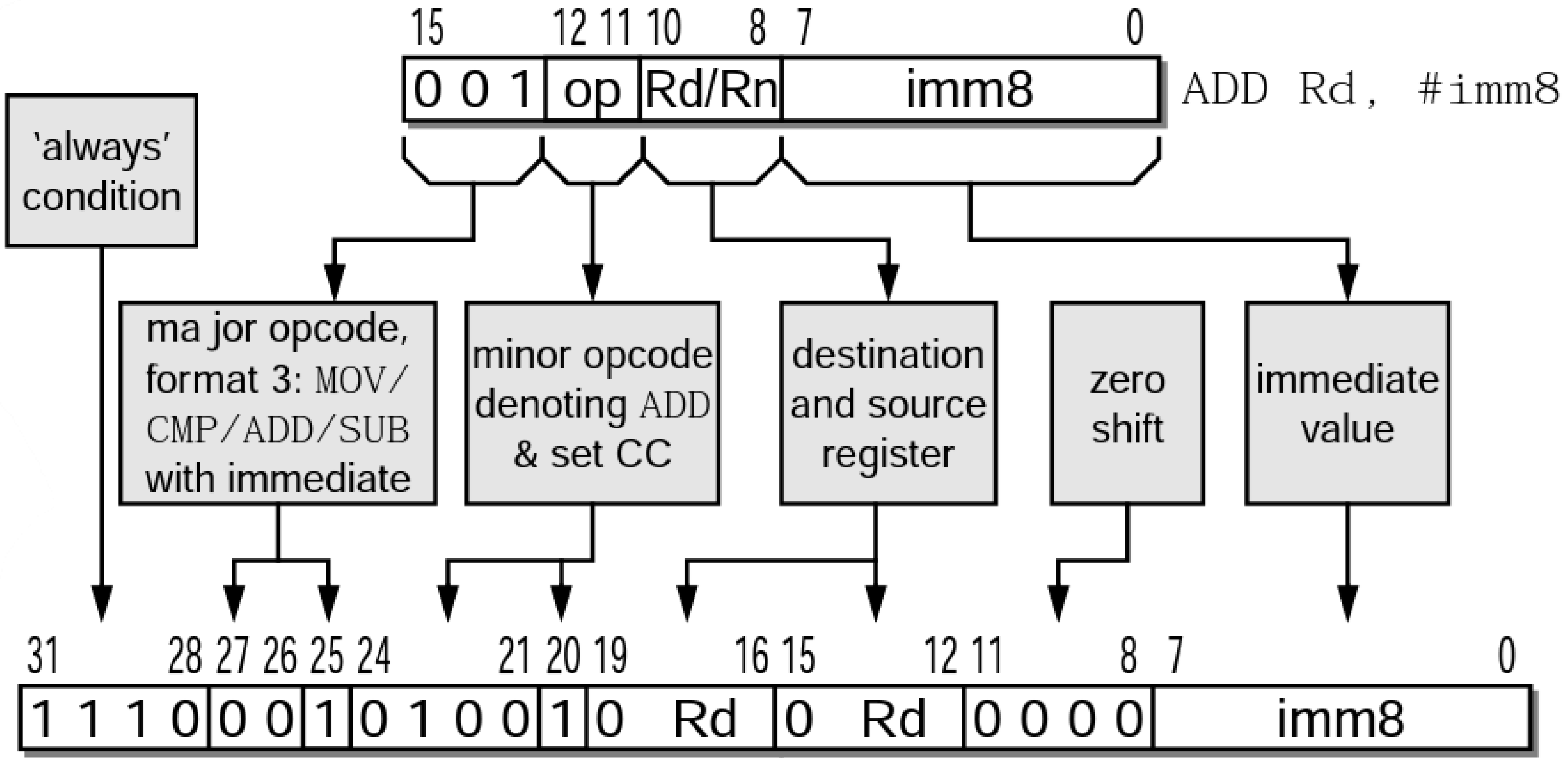
Thumb Branch Instruction

- These are similar to ARM instructions except:
  - offsets are scaled to half-word, not word
  - range is reduced to fit into 16 bits
  - BL works in two stages:

```
H=0: LR := PC + signextend(offset << 12)
H=1: PC := LR + (offset << 1)
LR := oldPC + 3
```

  - the assembler generates both halves
  - LR bit[0] is set to facilitate return via BX

Thumb - ARM instruction mapping



THUMB APPLICATIONS

- Thumb code properties:
  - 70% of the size of ARM code
    - 30% less external memory power
    - 40% more instructions
  - With 32-bit memory:
    - ARM code is 40% faster than Thumb code
  - With 16-bit memory:
    - Thumb code is 45% faster than ARM code

Thumb Applications

- For the best performance:
  - use 32-bit memory and ARM code
- For best cost and power-efficiency:
  - use 16-bit memory and Thumb code
- In a typical embedded system:
  - use ARM code in 32-bit on-chip memory for small speed-critical routines
  - use Thumb code in 16-bit off-chip memory for large non-critical control routines