The Thumb Instruction Set

Peng-Sheng Chen

Fall, 2017

Outline

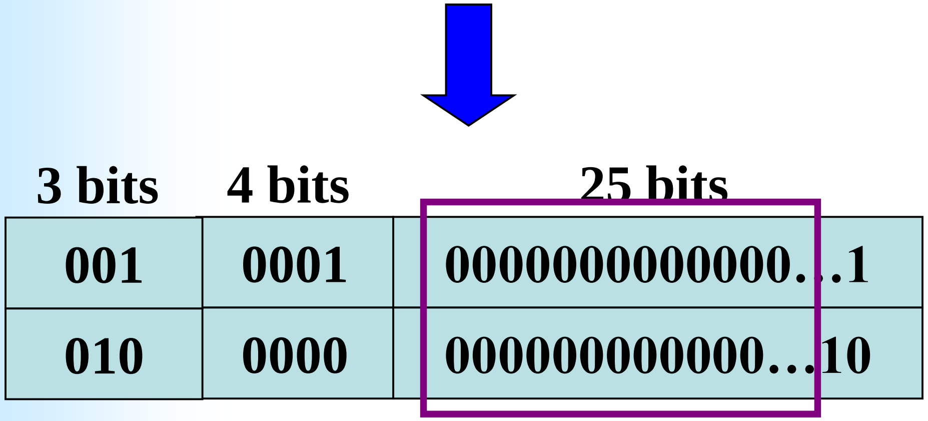
- Introduction
- Branch instruction
- Software interrupt instruction
- Data processing instructions
- Hi register operations
- Data transfer instructions

Outline

- **Introduction**
- Branch instruction
- Software interrupt instruction
- Data processing instructions
- Hi register operations
- Data transfer instructions

故事是這樣發生的...

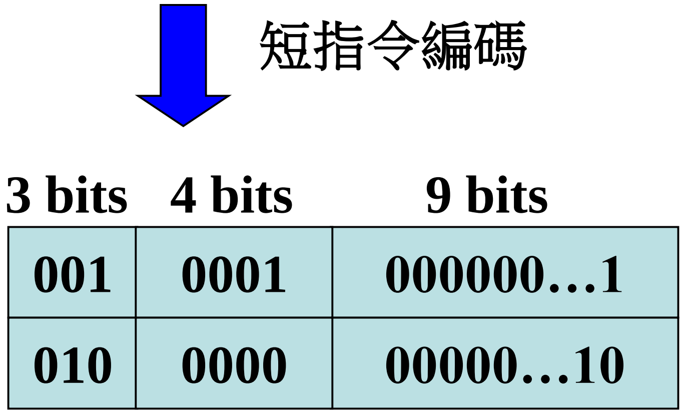
```
ADD   r1,#1
MOV   r0,#2
```



3 bits	4 bits	25 bits
001	0001	0000000000000...1
010	0000	0000000000000...10

Code size= 8 bytes
(很多空間浪費了...)

短指令編碼



3 bits	4 bits	9 bits
001	0001	000000...1
010	0000	00000...10

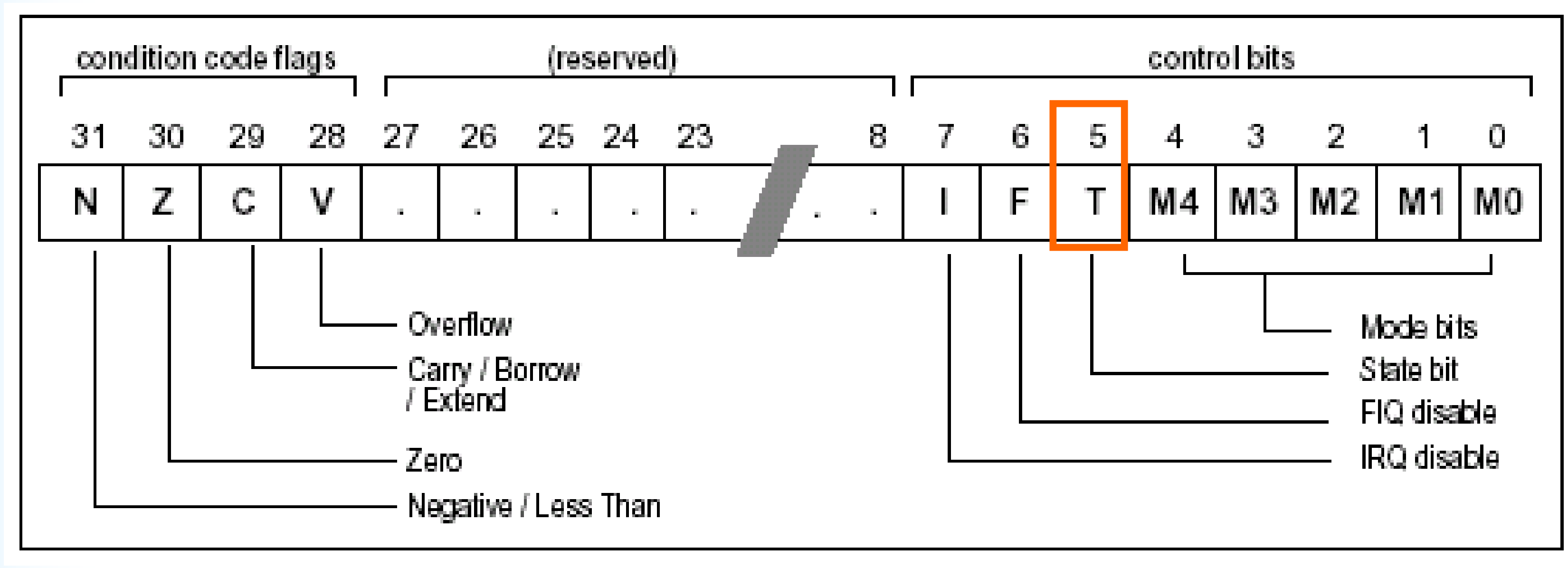
Code size= 4 bytes
(程式碼大小是原來的一半)

Thumb Instruction Set

- **Thumb addresses code density**
  - A compressed form of a subset of the ARM instruction set
- **Thumb maps onto ARMs**
  - Dynamic decompression in an ARM instruction pipeline
  - Instructions execute as standard ARM instructions within the processor
- **Thumb is not a complete architecture**
- **Thumb is fully supported by ARM development tools**
- **Design for processor / compiler, not for programmer**

The Thumb Bit in the CPSR (1)

- The interpretation of the instruction stream
  - Bit 5 of the CPSR (the **T** bit)
  - If **T** is set => interpret as 16-bit Thumb inst.



The Thumb Bit in the CPSR (2)

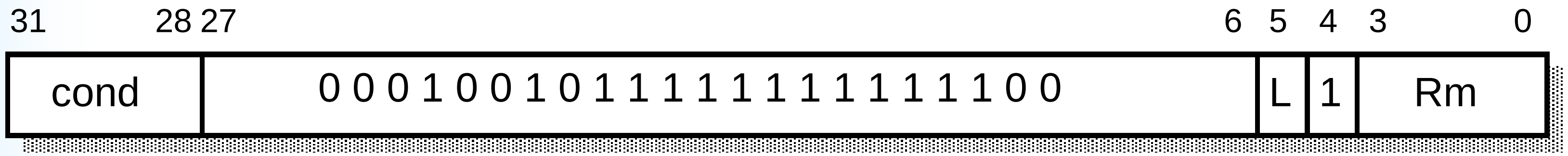
- **Thumb entry**
  - ARM cores start up, after reset, executing ARM instructions
  - Thumb is executed normally by using **BX** (**Branch and eXchange instruction**)
  - Switching flushes the instruction pipeline
  - Other ways change from ARM to Thumb code
    - Exception return
- **Thumb exit**
  - Restore CPSR (explicitly or implicitly)

The Thumb Bit in the CPSR (3)

- If Rm[0] is 1, switch to Thumb
- If Rm[0] is 0, continue ARM

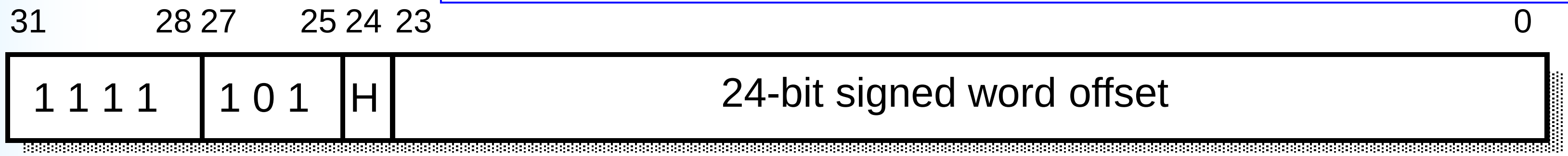
(1) `BX|BLX Rm`

BX|BLX{cond} Rm (absolute address)

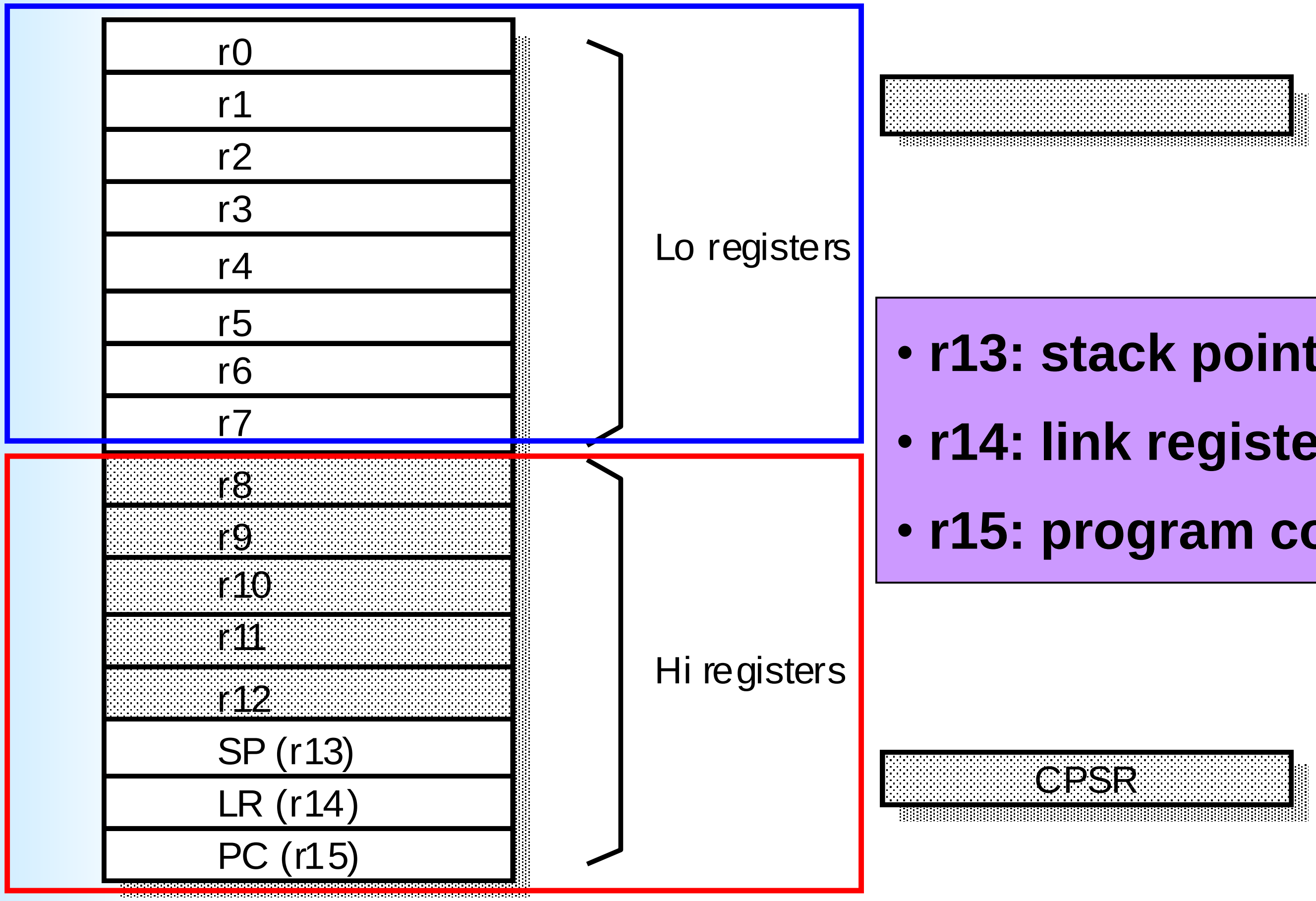


(2) `BLX label`

BLX label (unconditional, PC-relative address)
Always causes a change to Thumb state



Thumb Accessible Registers



- **r13: stack pointer (SP)**
- **r14: link register (LR)**
- **r15: program counter (PC)**

Thumb-ARM Similarities

- **Thumb inherits many ARM's properties**
  - The load-store architecture with data processing, data transfer and control flow instructions
  - Support for 8-bit byte, 16-bit half-word, and 32-bit word data types
  - A 32-bit unsegmented memory

Thumb-ARM Differences (1)

- All Thumb instructions are 16-bits long
  - ARM instructions are 32-bits long
- Most Thumb instructions are executed **unconditionally**
  - All ARM instructions are executed conditionally

Thumb-ARM Differences (2)

- Many Thumb data processing instructions use a **2-address** format (the destination register is the same as one of the source registers)
  - **ARM use 3-address format**
- Thumb instruction are **less regular** than ARM instruction formats, as a result of the dense encoding

Thumb Applications

- **Thumb properties**
  - Thumb requires **70%** space of the ARM code
  - Thumb uses **40%** more instructions than the ARM code

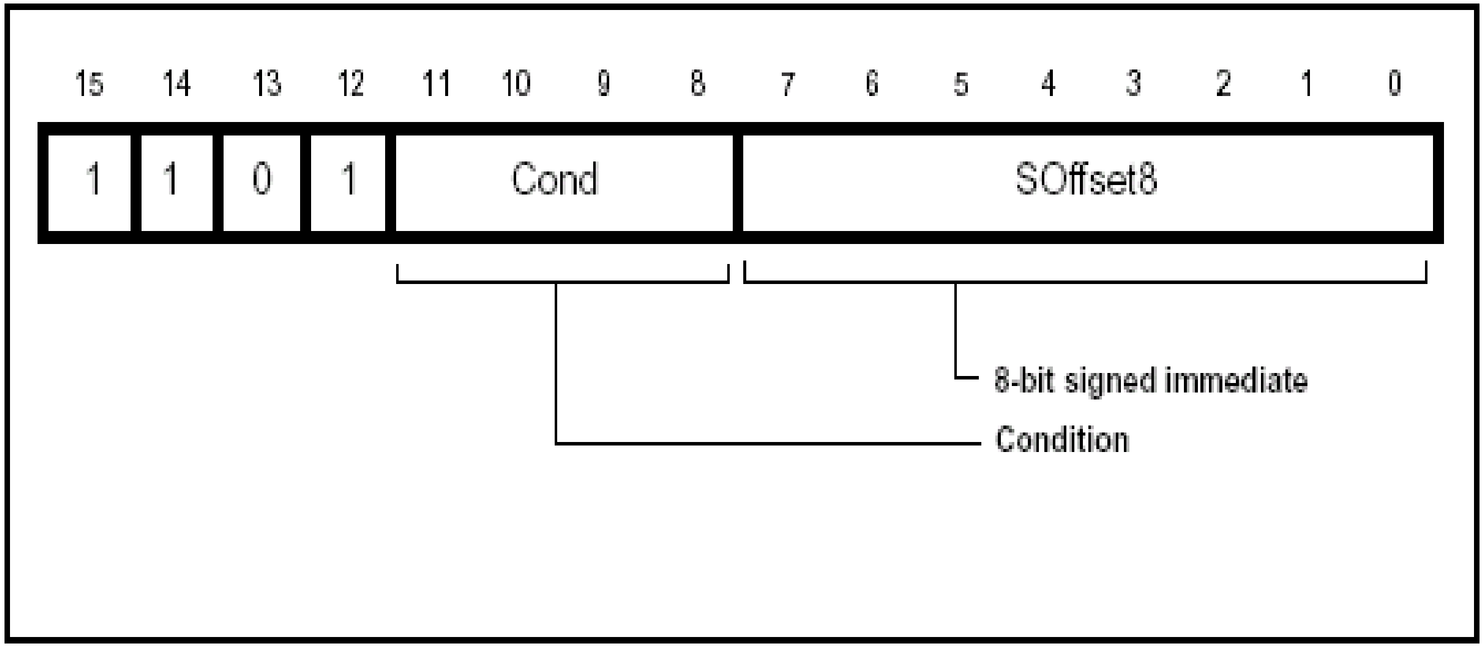
Outline

- Introduction
- **Branch instruction**
- Software interrupt instruction
- Data processing instructions
- Hi register operations
- Data transfer instructions

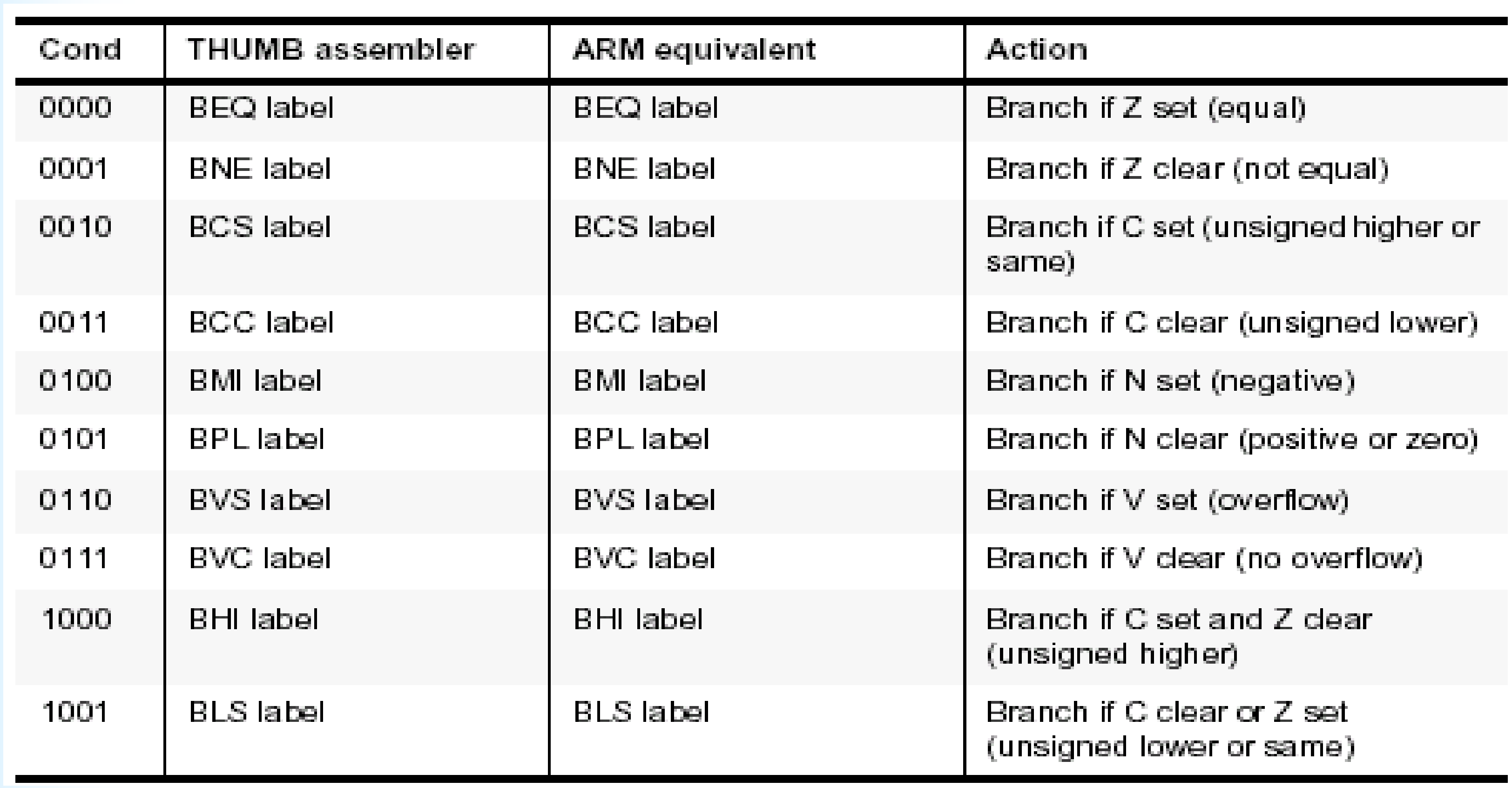
Thumb Branch Instructions (1)

Conditional Branch (pc-relative)

Format: B<cond> <label>



Assembler Syntax



Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)

Assembler Syntax (cont'd)

Cond	THUMB assembler	ARM equivalent	Action
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

Note While label specifies a full 9-bit two's complement address, this must always be halfword-aligned (ie with bit 0 set to 0) since the assembler actually places label >> 1 in field SOffset8.

Note Cond = 1110 is undefined, and should not be used.
Cond = 1111 creates the SWI instruction

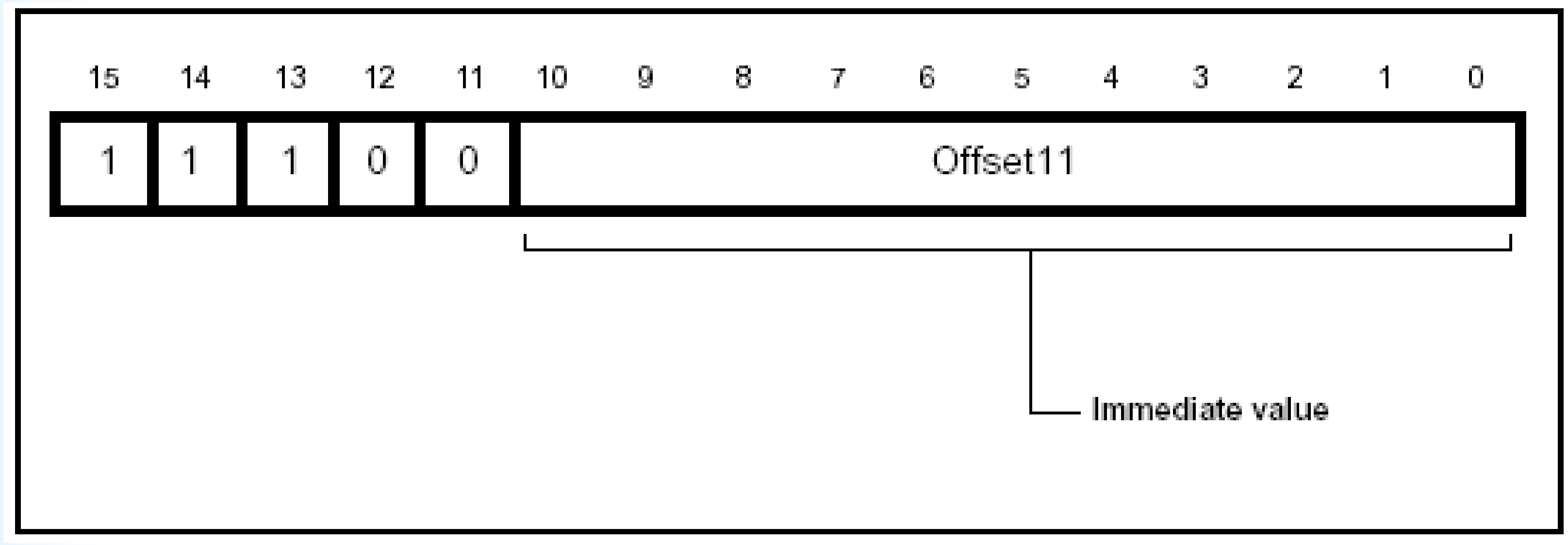
Example (Conditional Branch)

```
      CMP    r0, #45      ; compare r0 and #45
      BGT    LABEL        ; if r0 > #45, then
                          ; branch to 'LABEL'
                          ; Thumb opcode will contain
                          ; the number of halfwords to
                          ; offset

LABEL ...                 ; Must be halfword aligned
```

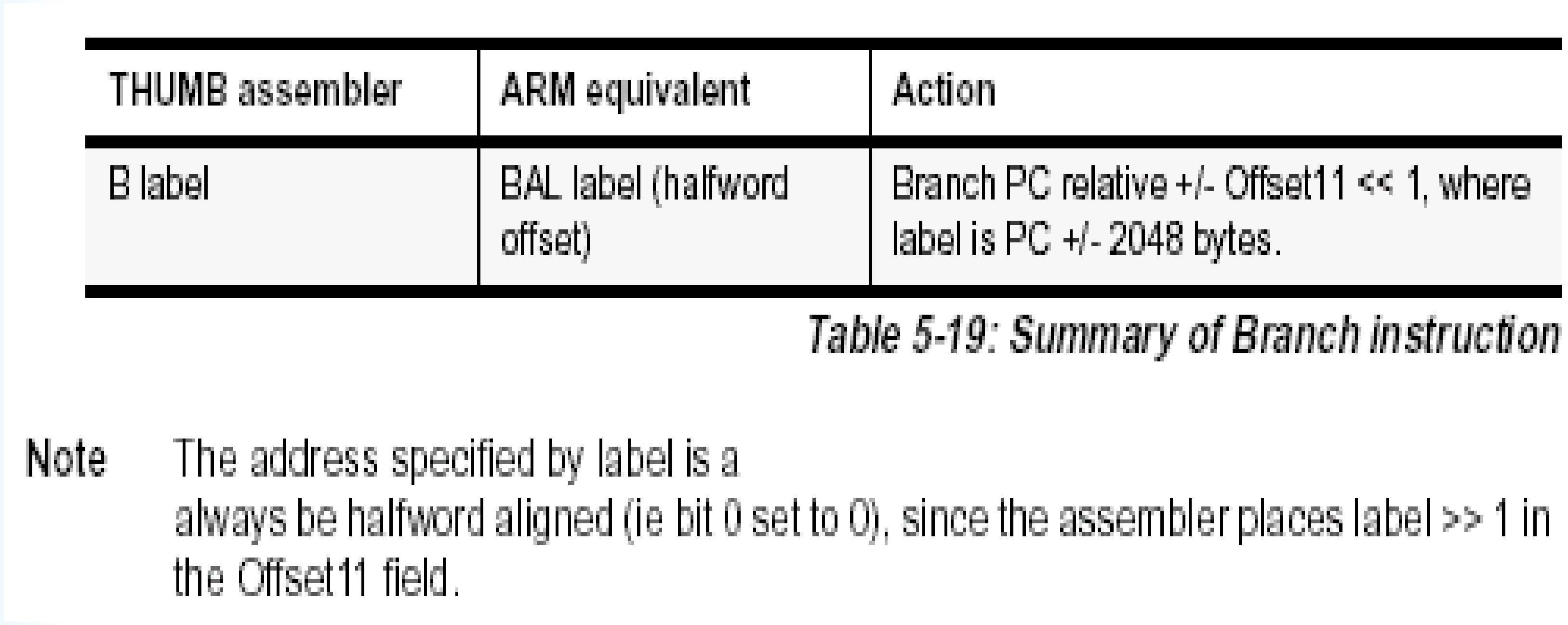
Thumb Branch Instructions (2)

Unconditional Branch

Format: B <label>



Assembler Syntax



THUMB assembler	ARM equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

Table 5-19: Summary of Branch instruction

Note The address specified by label is a always be halfword aligned (ie bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

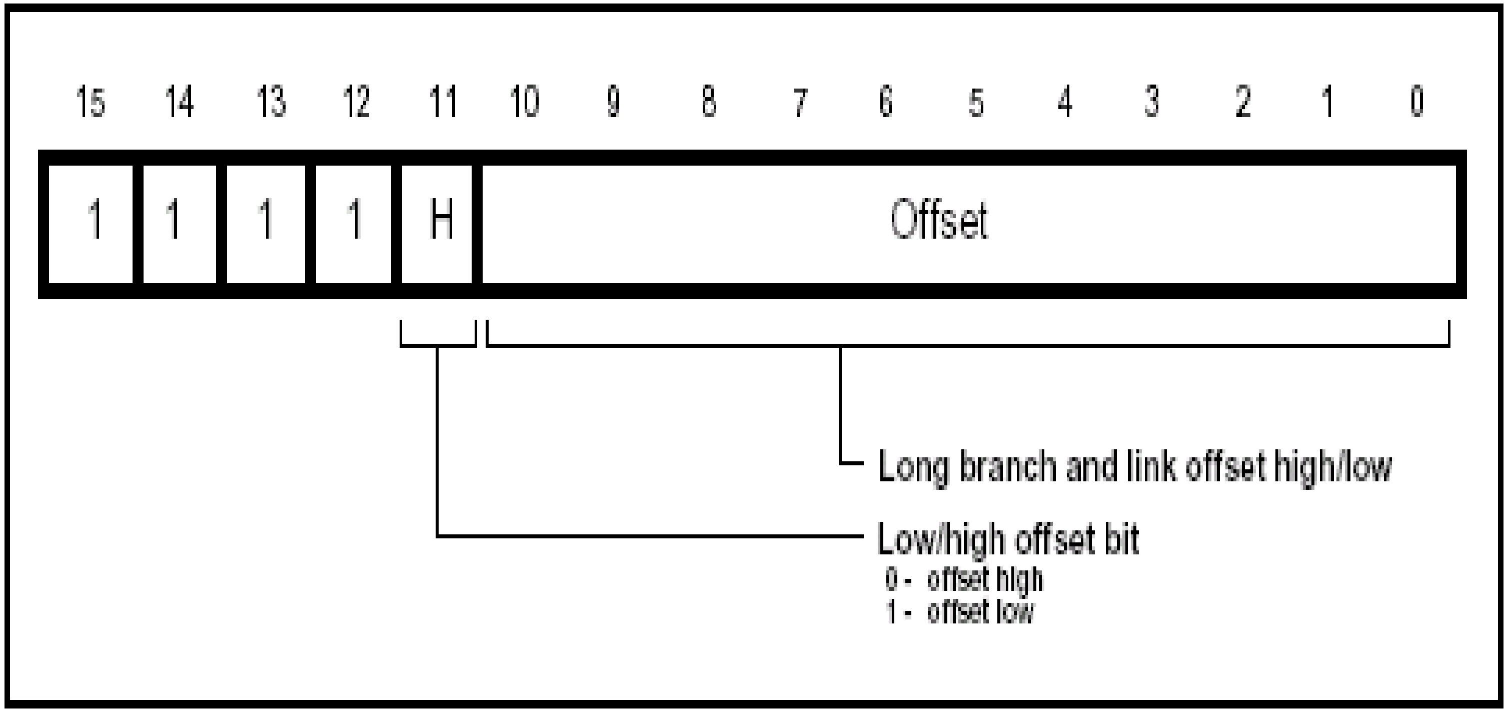
Example (Unconditional Branch)

```
      B jimmy            ; Branch to 'jimmy'.
       ...               ; Note that the THUMB opcode will
                         ; contain the number of halfwords
                         ; to offset.
jimmy    ...             ; Must be halfword aligned.
```

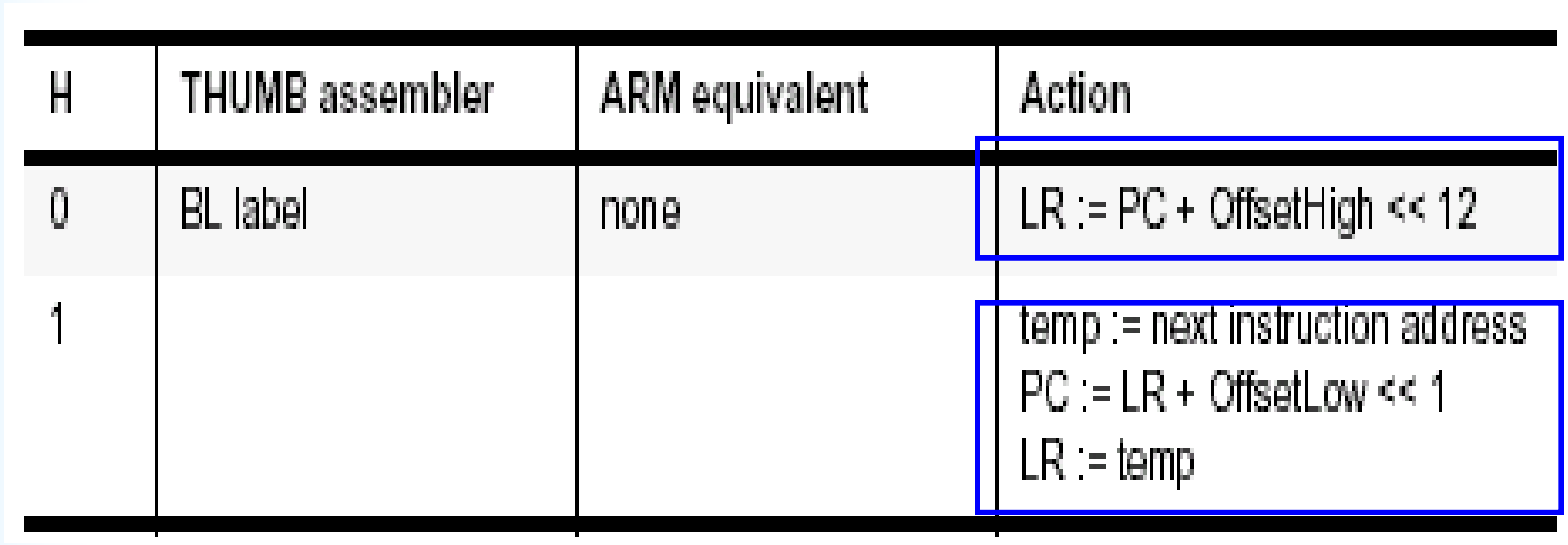
Thumb Branch Instructions (3)

Long Branch with Link

Format: BL <label>



Assembler Syntax



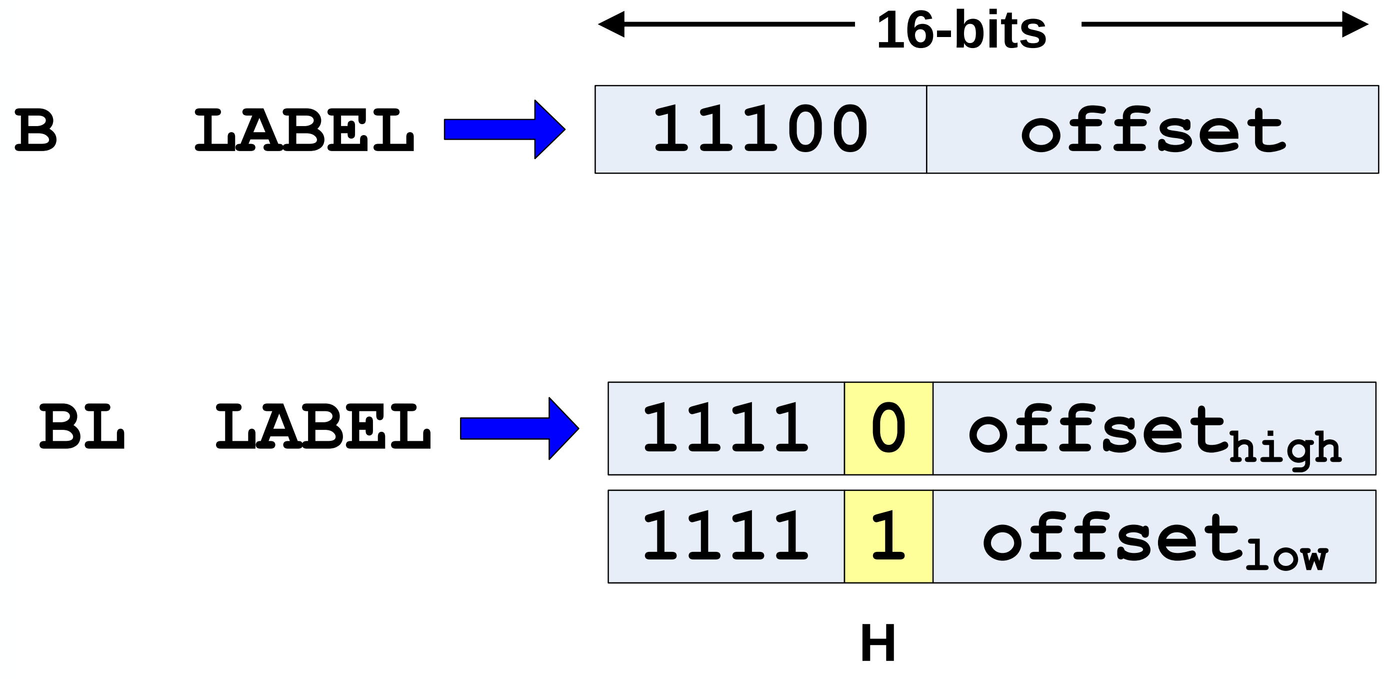
H	THUMB assembler	ARM equivalent	Action
0	BL label	none	LR := PC + OffsetHigh << 12

1			temp := next instruction address
			PC := LR + OffsetLow << 1
			LR := temp

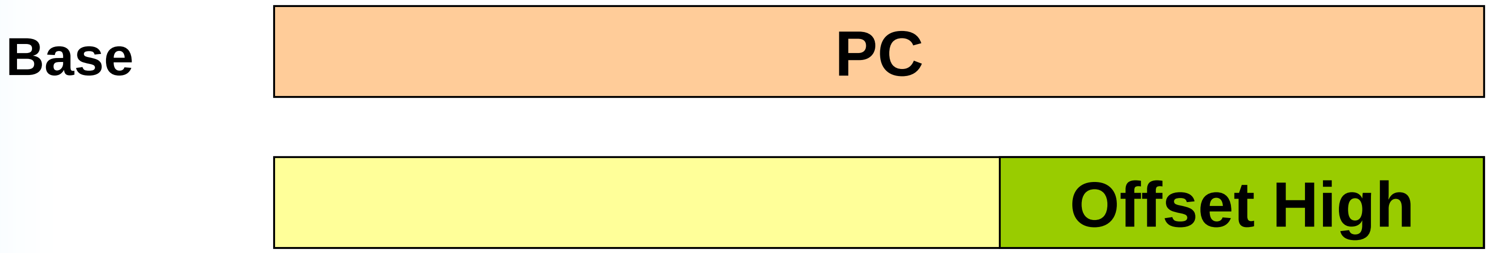
1

2

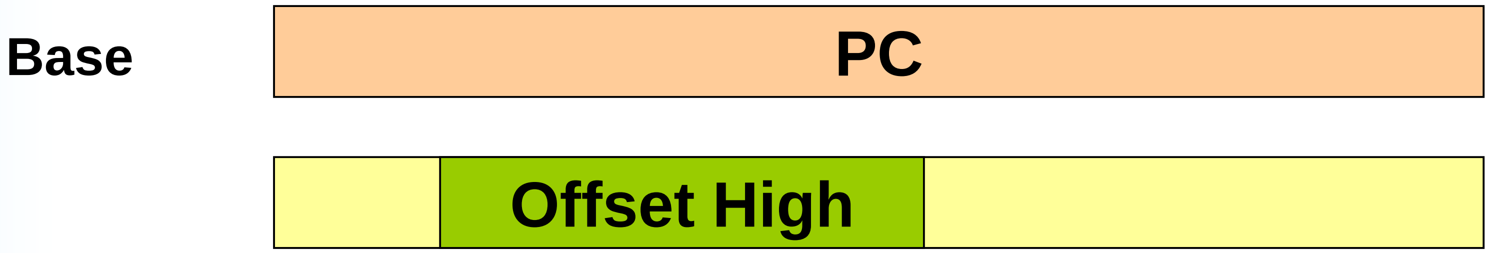
Two-Instruction Compound Instruction (1)



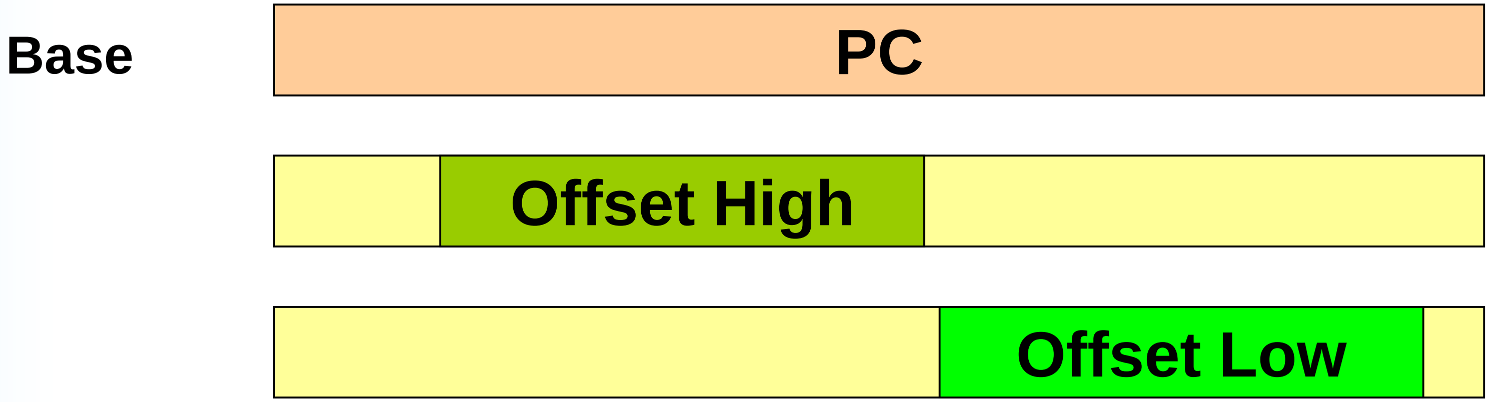
Two-Instruction Compound Instruction (2)



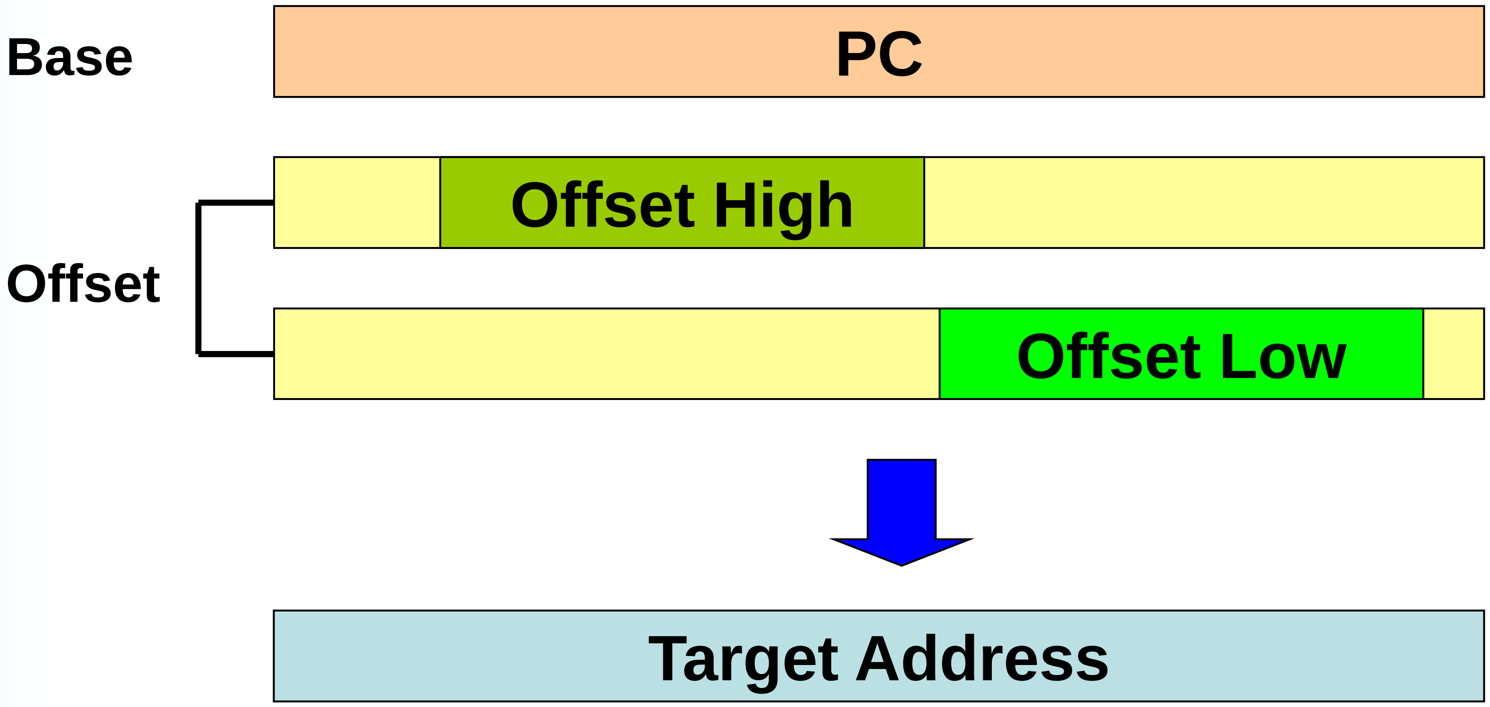
Two-Instruction Compound Instruction (2)



Two-Instruction Compound Instruction (2)

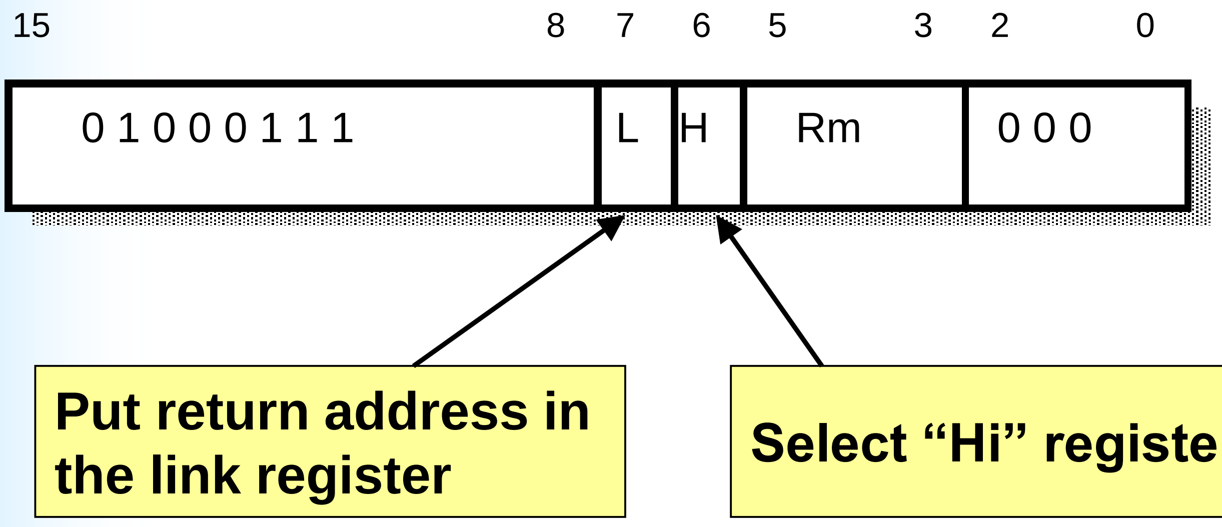


Two-Instruction Compound Instruction (2)



Thumb Branch Instructions (4)

- If Rm[0] is 1, switch to Thumb
- If Rm[0] is 0, continue ARM



`B{L}X  Rm`

Thumb Branch Instructions Summary of Assembler Format

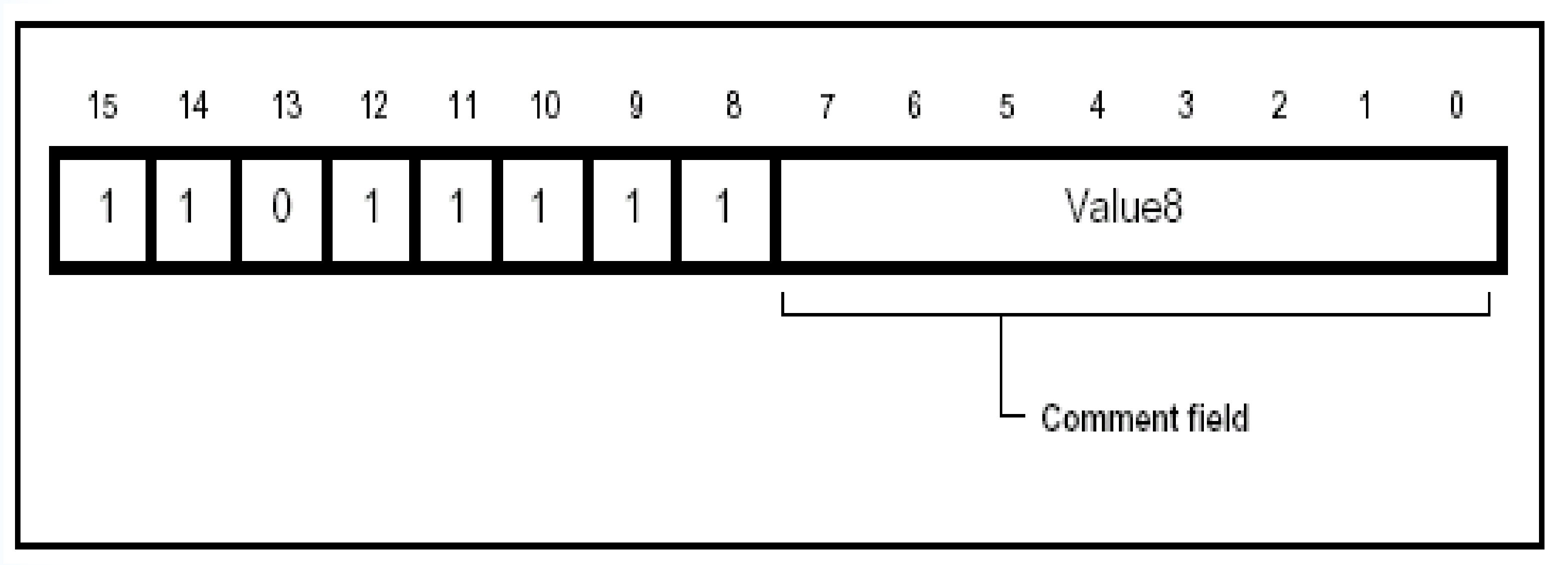
- B<cond> <label> ; Thumb target
- B <label> ; Thumb target
- BL <label> ; Thumb target
- BLX <label> ; Thumb target
- B{L}X Rm ; ARM or Thumb target

Outline

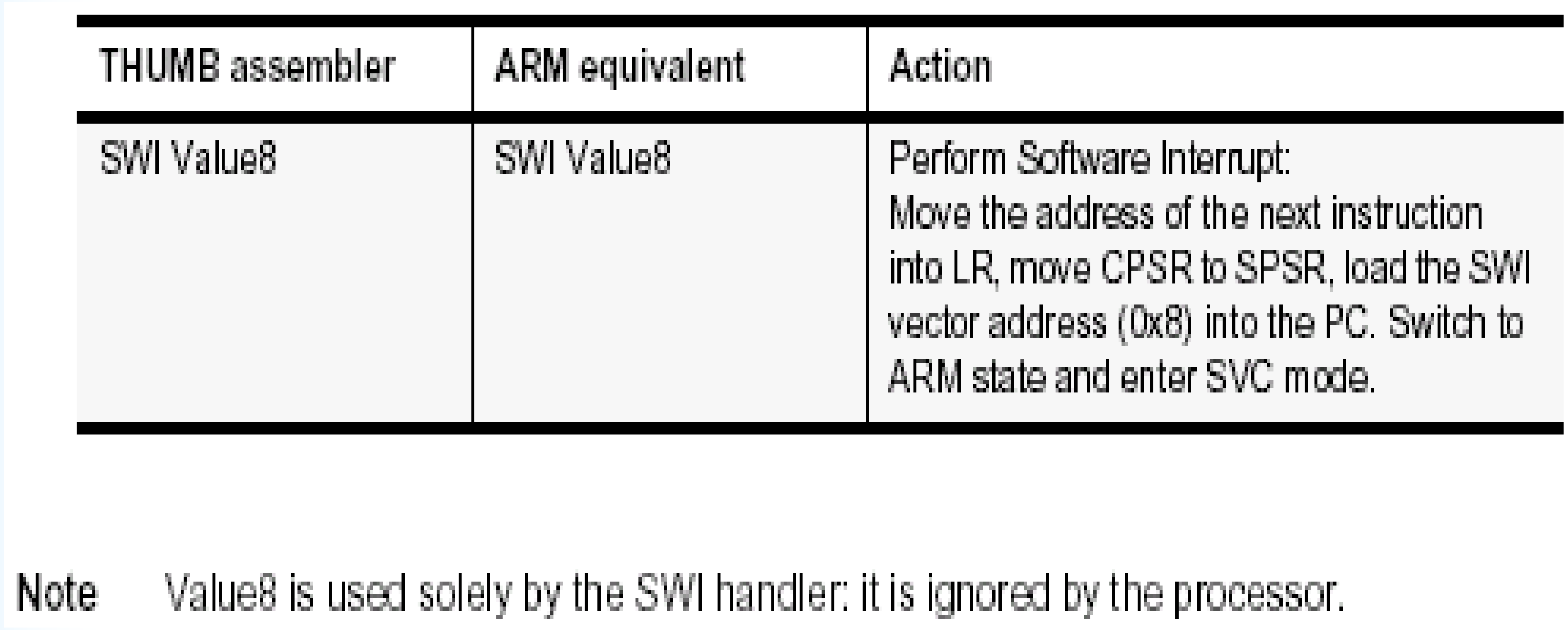
- Introduction
- Branch instruction
- **Software interrupt instruction**
- Data processing instructions
- Hi register operations
- Data transfer instructions

Thumb Software Interrupt

Format: SWI <8-bit immediate>



Assembler Syntax



THUMB assembler	ARM equivalent	Action
SWI Value8	SWI Value8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

Note Value8 is used solely by the SWI handler: it is ignored by the processor.

Example

```
SWI 18            ; Take the software interrupt exception.
                  ; Enter Supervisor mode with 18 as the
                  ; requested SWI number.
```

Outline

- Introduction
- Branch instruction
- Software interrupt instruction
- **Data processing instructions**
- Hi register operations
- Data transfer instructions

Thumb Data Processing Instructions

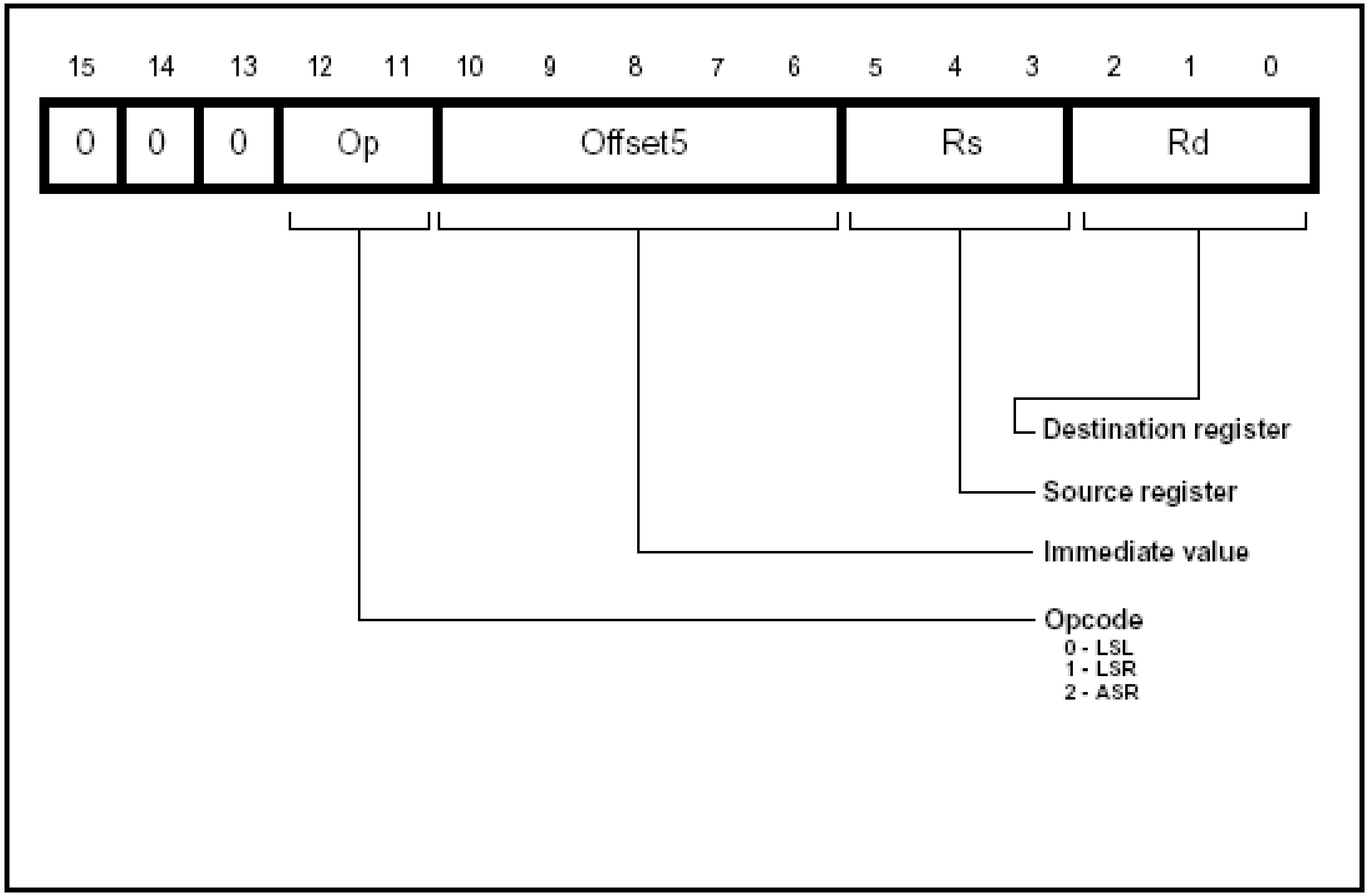
- Move shifted register
- Add/subtract
- Move/compare/add/subtract immediate
- ALU operations
- Add PC/SP
- Add/Sub offset to stack pointer

Thumb Data Processing Instructions

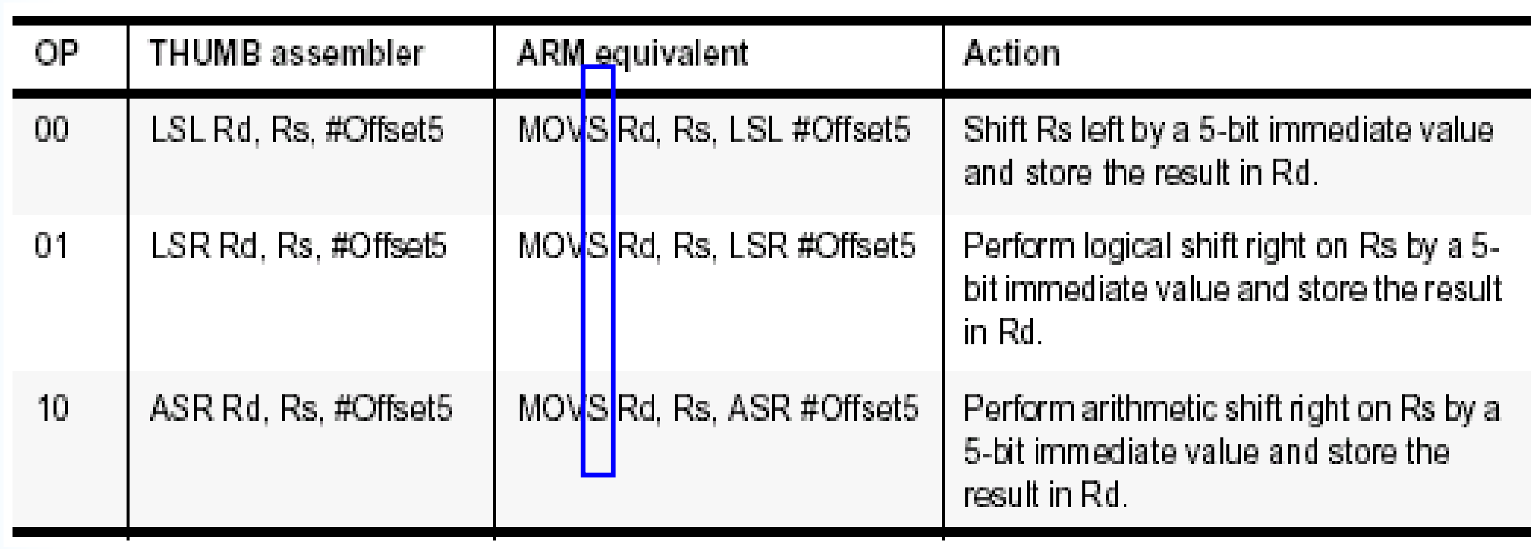
- **Move shifted register**
- Add/subtract
- Move/compare/add/subtract immediate
- ALU operations
- Add PC/SP
- Add/Sub offset to stack pointer

Move Shifted Register

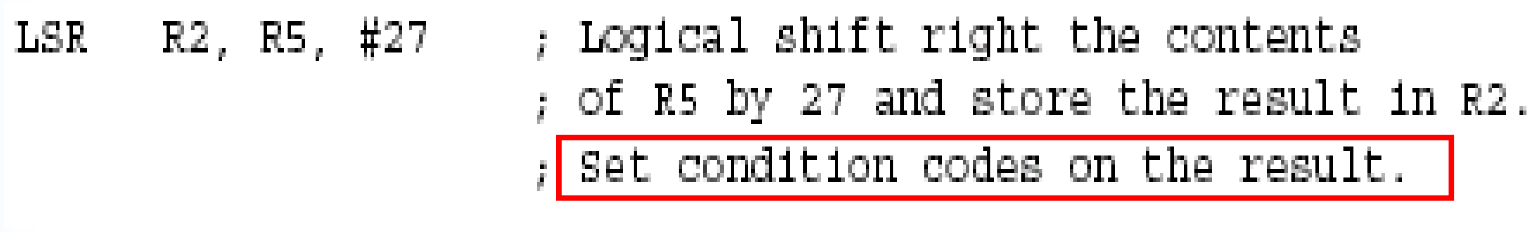
Format: LSL|LSR|ASR   Rd, Rs, #shift



Assembler Syntax and Example



OP	THUMB assembler	ARM equivalent	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.



```
LSR    R2, R5, #27      ; Logical shift right the contents
                        ; of R5 by 27 and store the result in R2.
                        ; Set condition codes on the result.
```

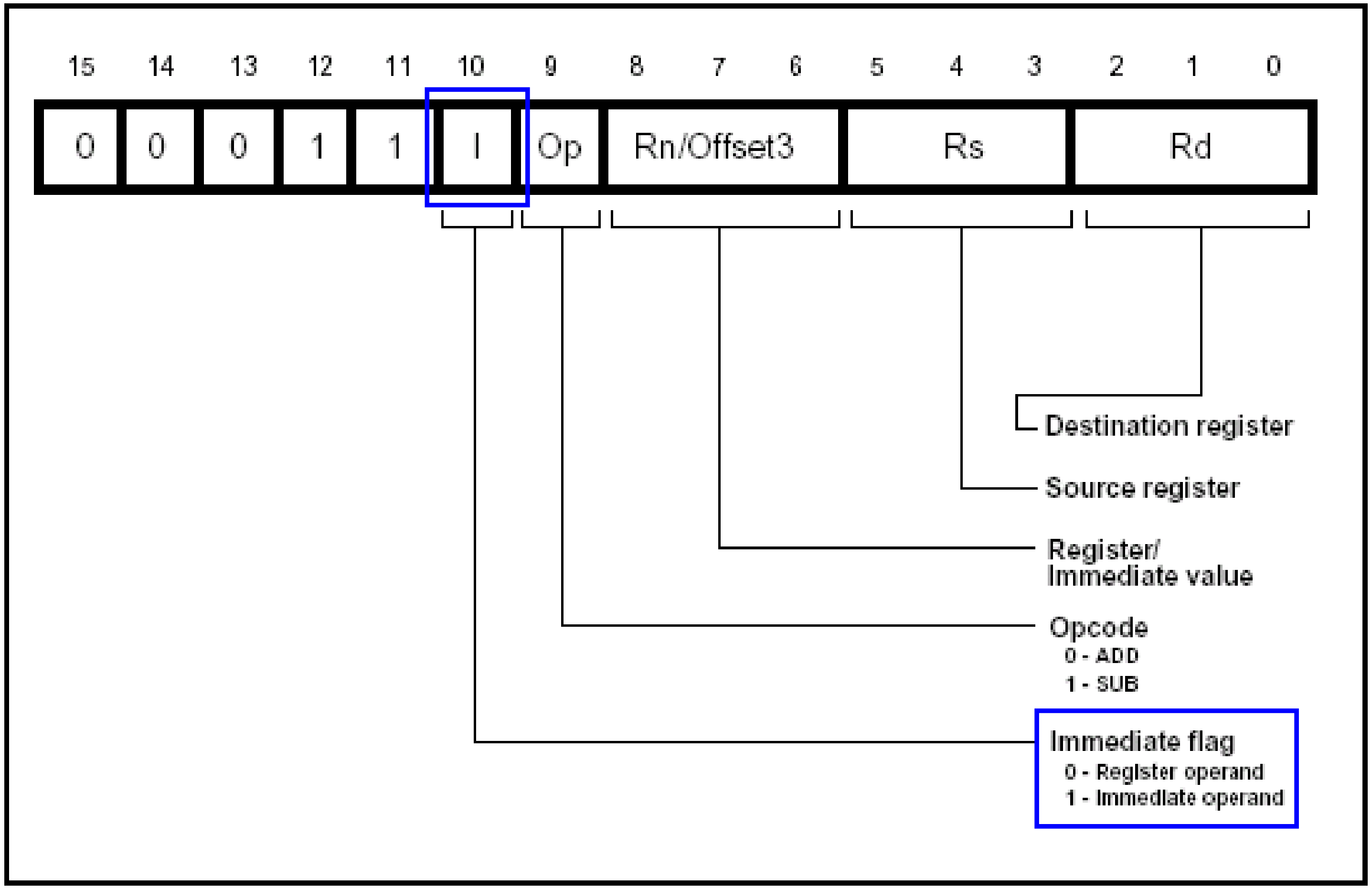
Thumb Data Processing Instructions

- Move shifted register
- **Add/subtract**
- Move/compare/add/subtract immediate
- ALU operations
- Add PC/SP
- Add offset to stack pointer

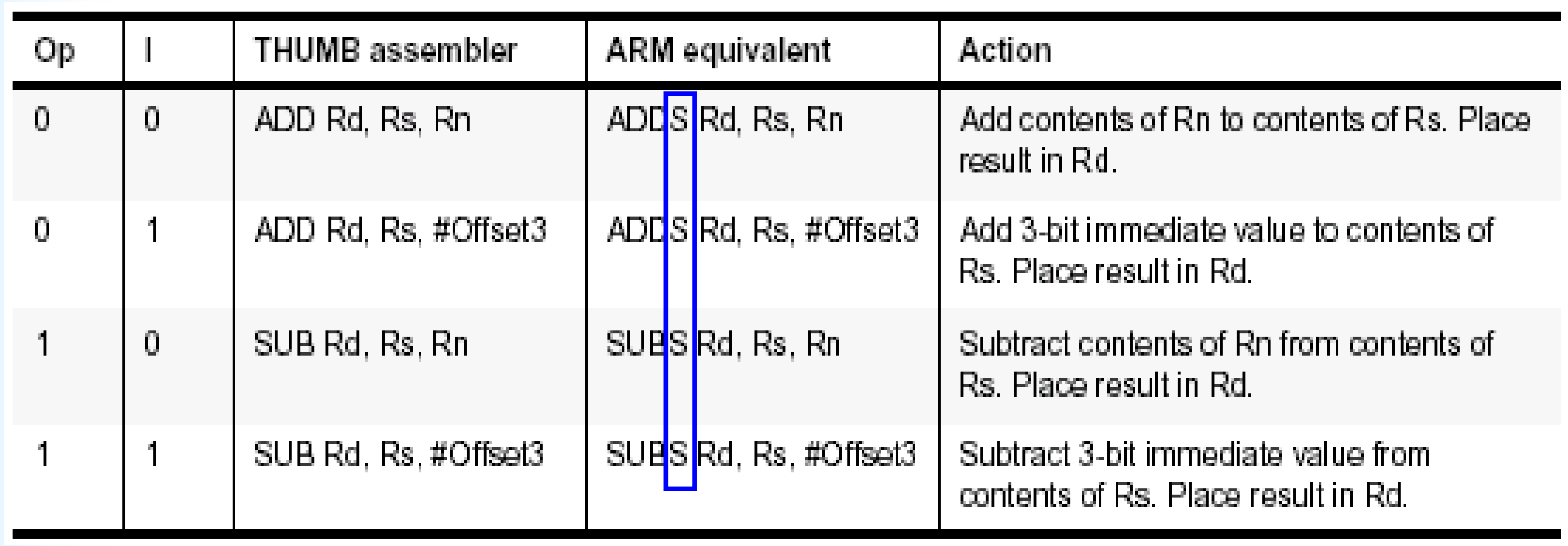
Add and Subtract

Format: ADD|SUB Rd, Rs, Rn

ADD|SUB Rd, Rs, #imm3



Assembler Syntax and Example



Op	I	THUMB assembler	ARM equivalent	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

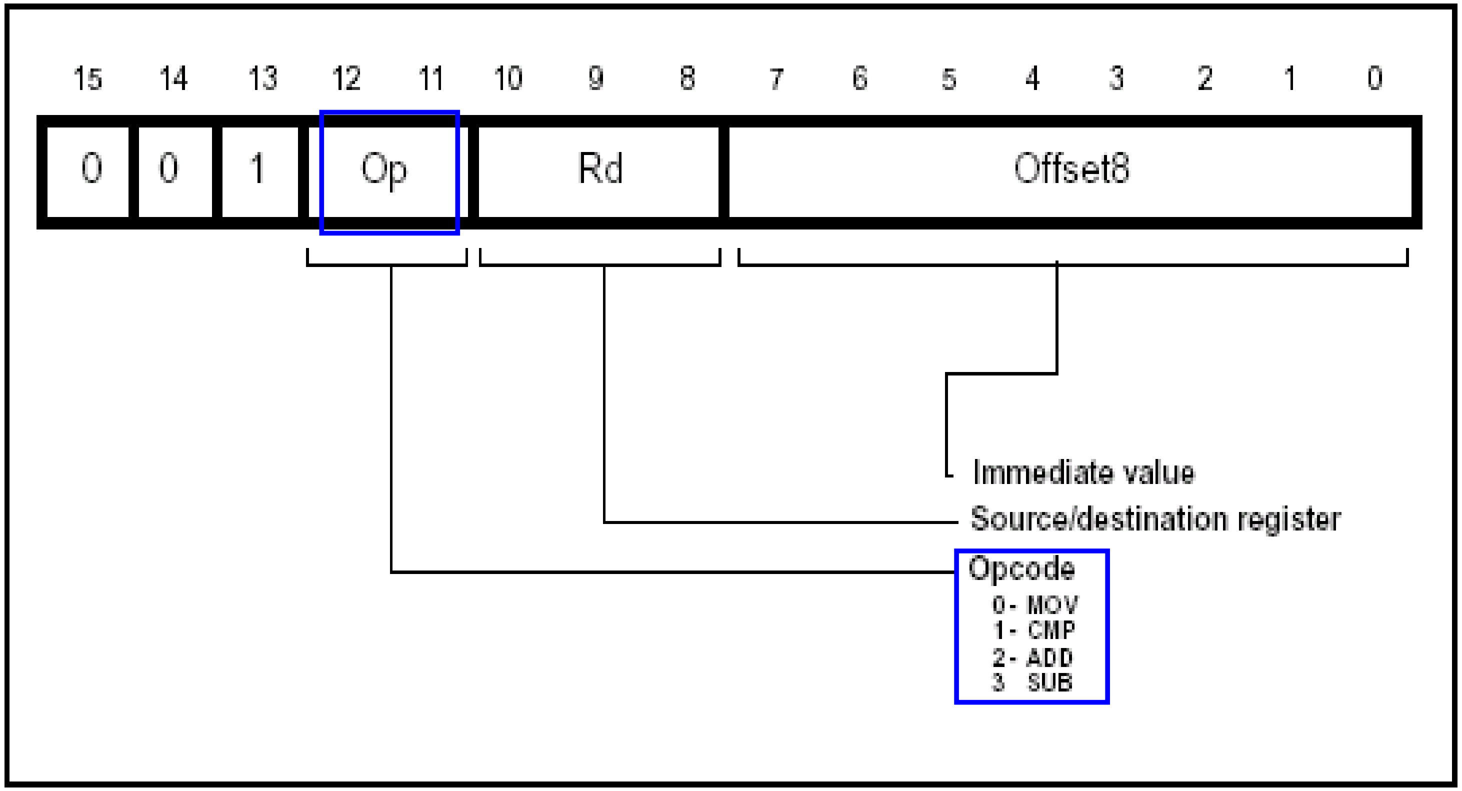
```
ADD   R0, R3, R4      ; R0 := R3 + R4 and set condition codes on
                      ; the result.

SUB   R6, R2, #6      ; R6 := R2 - 6 and set condition codes.
```

Thumb Data Processing Instructions

- Move shifted register
- Add/subtract
- **Move/compare/add/subtract immediate**
- ALU operations
- Add PC/SP
- Add offset to stack pointer

Move/Compare/Add/Subtract Immediate

Format: <Op>   Rd, #imm8



Assembler Syntax and Example

Op	THUMB assembler	ARM equivalent	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

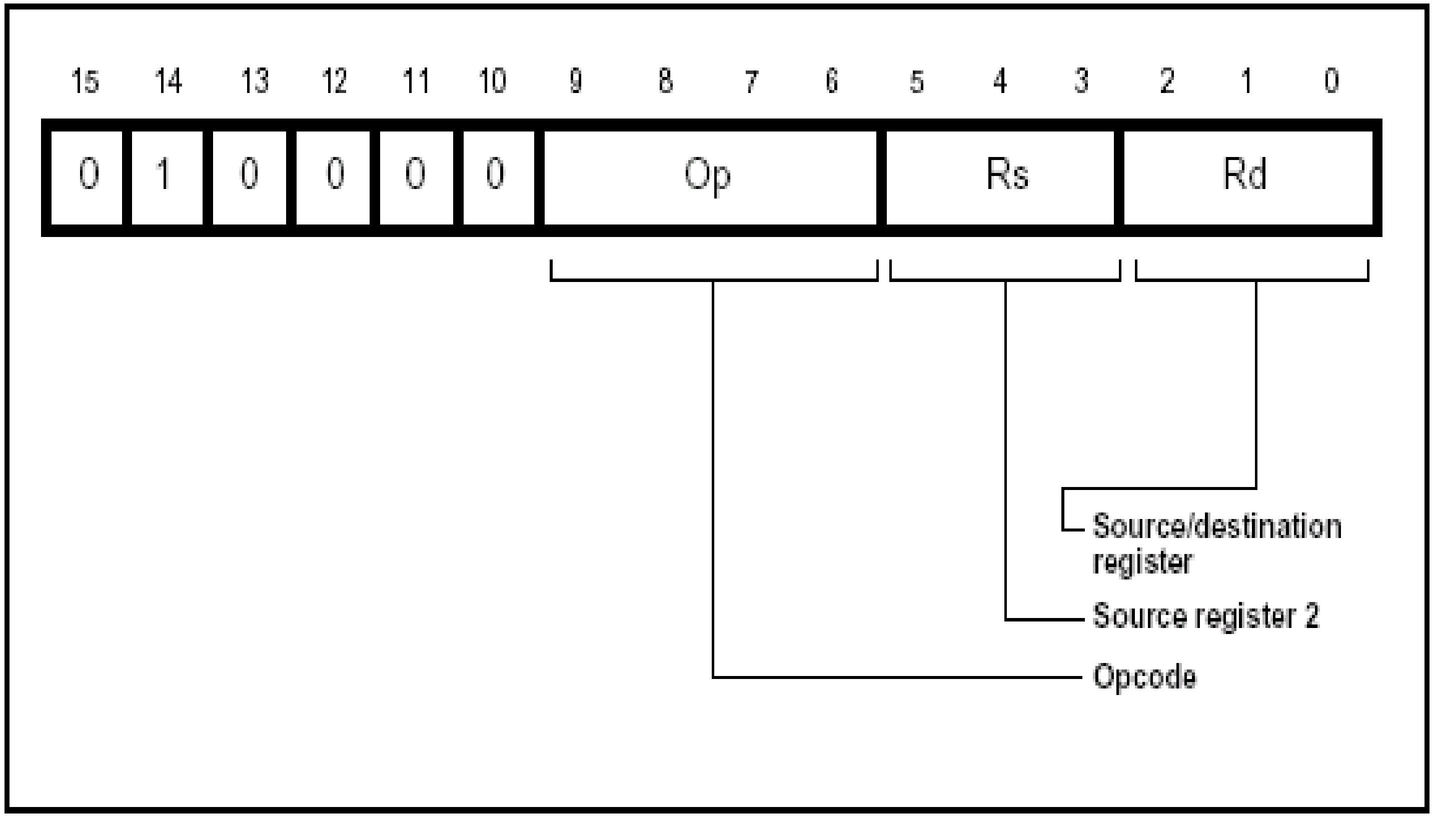
```
MOV   R0, #128       ; R0 := 128 and set condition codes

CMP   R2, #62        ; Set condition codes on R2 - 62

ADD   R1, #255       ; R1 := R1 + 255 and set condition
                     ; codes

SUB   R6, #145       ; R6 := R6 - 145 and set condition
                     ; codes
```

Thumb Data Processing Instructions

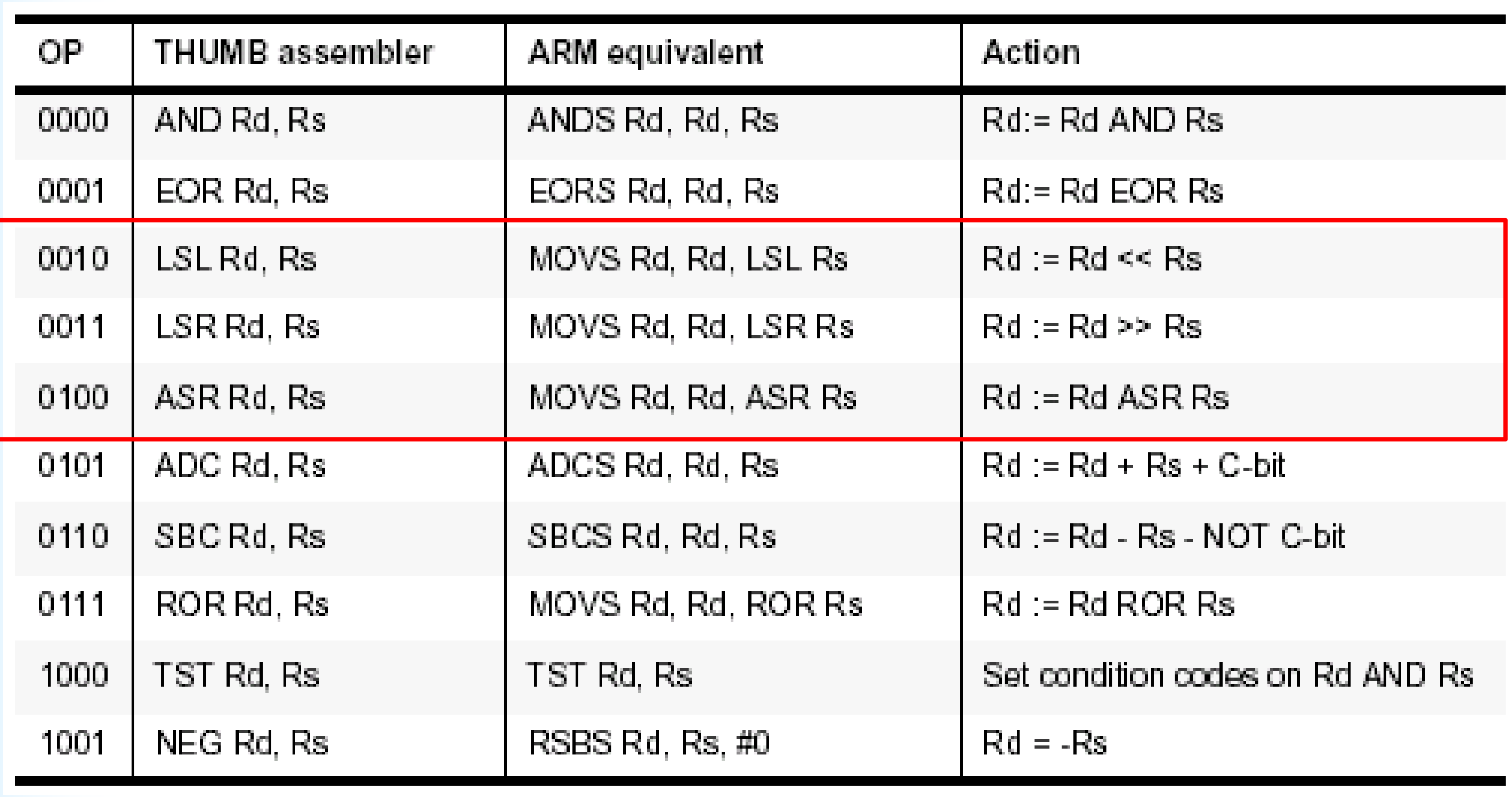
- Move shifted register
- Add/subtract
- Move/compare/add/subtract immediate
- **ALU operations**
- Add PC/SP
- Add offset to stack pointer

ALU Operations

Format: <Op>   Rd, Rs

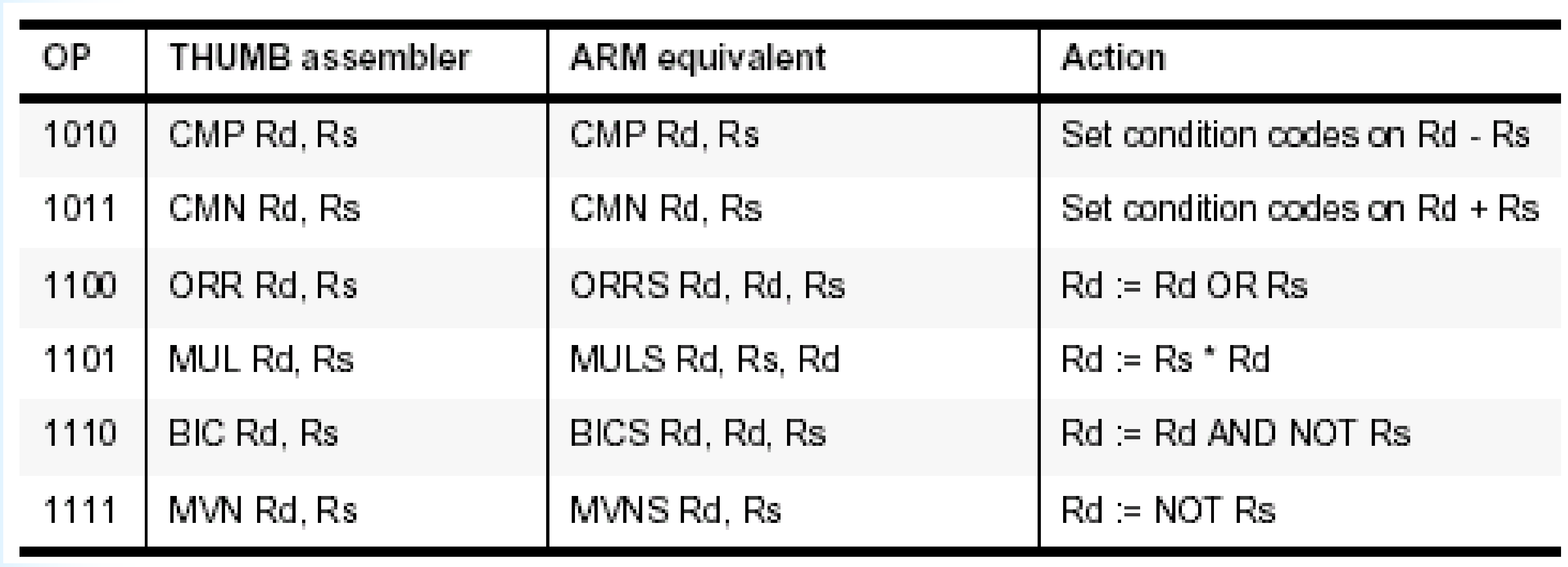


Assembler Syntax (1)



OP	THUMB assembler	ARM equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = -Rs

Assembler Syntax (2)



OP	THUMB assembler	ARM equivalent	Action
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

Example

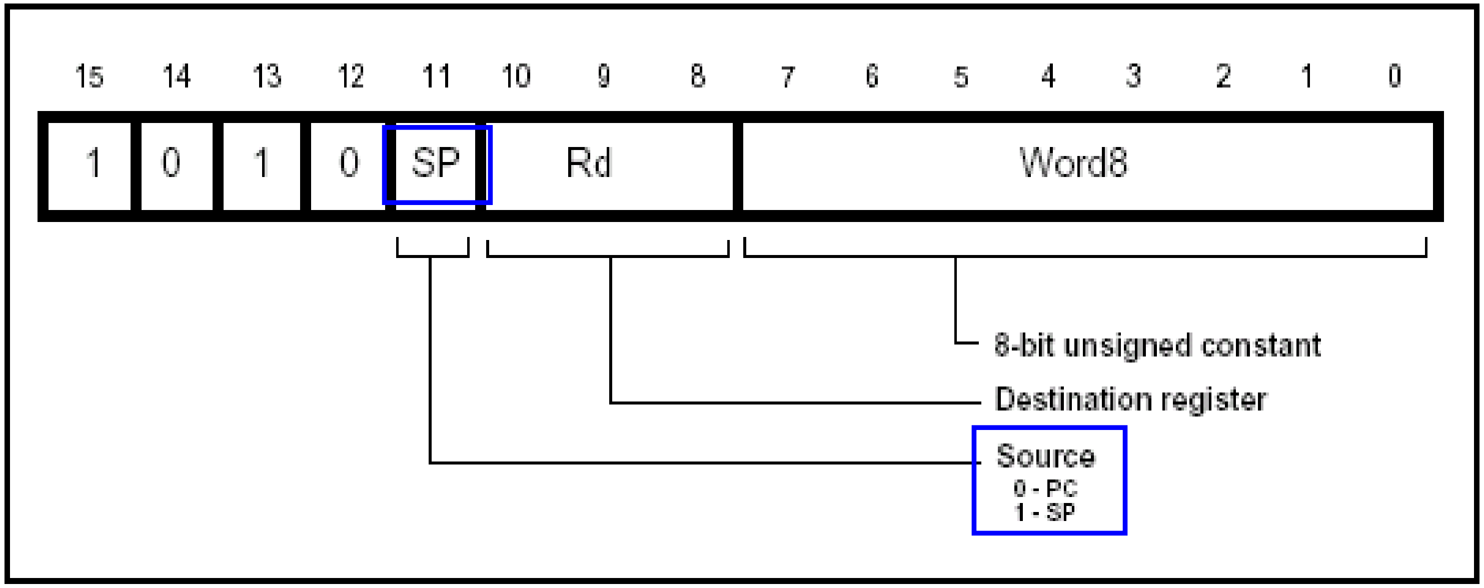
```
EOR    R3, R4    ; R3 := R3 EOR R4 and set condition codes

ROR    R1, R0    ; Rotate Right R1 by the value in R0, store
                 ; the result in R1 and set condition codes

NEG    R5, R3    ; Subtract the contents of R3 from zero,
                 ; store the result in R5. Set condition codes
                 ; ie R5 = -R3

CMP    R2, R6    ; Set the condition codes on the result of
                 ; R2 - R6

MUL    R0, R7    ; R0 := R7 * R0 and set condition codes
```

Thumb Data Processing Instructions

- Move shifted register
- Add/subtract
- Move/compare/add/subtract immediate
- ALU operations
- **Add PC/SP**
- Add offset to stack pointer

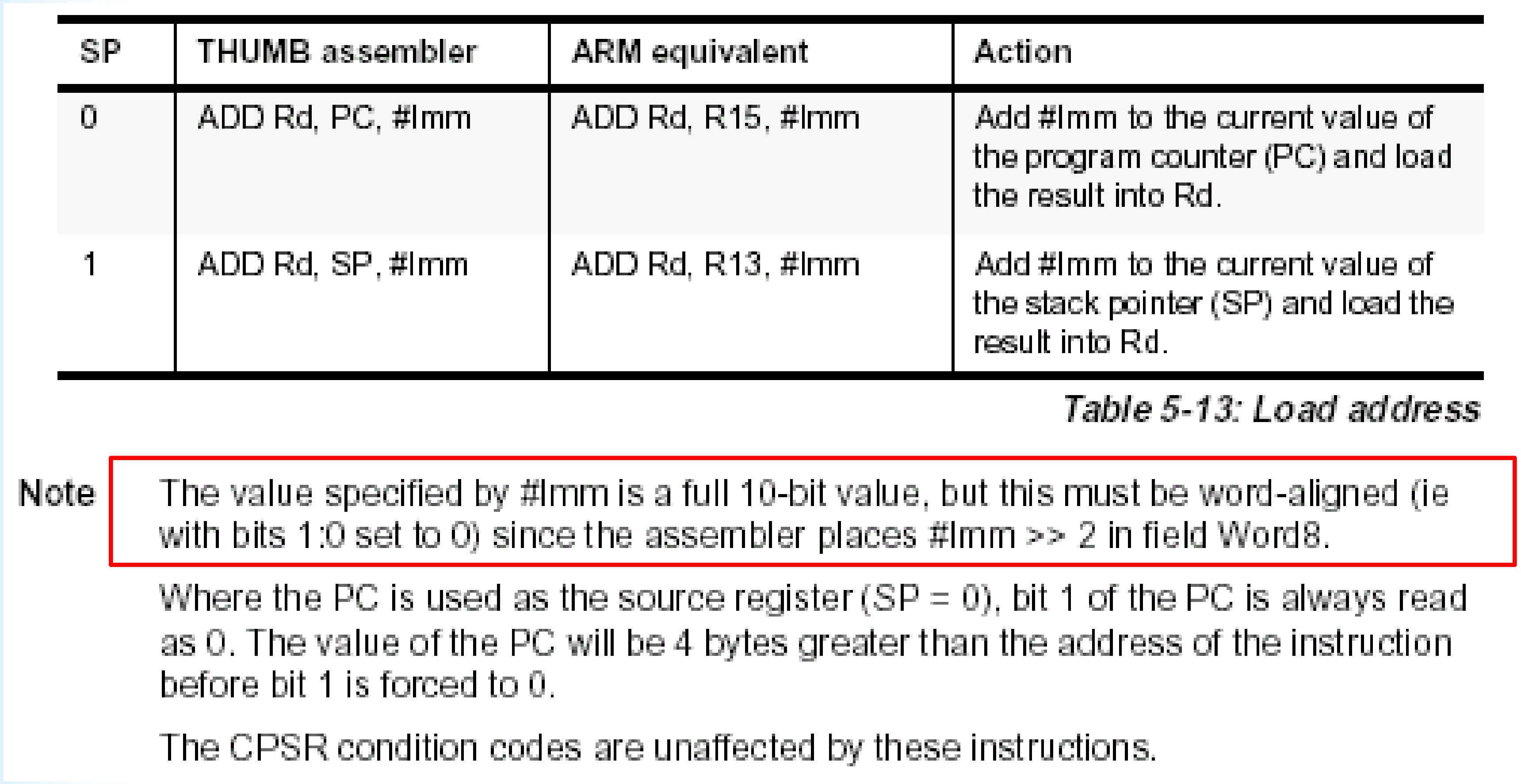
ADD PC / SP

Format: ADD   Rd, SP | PC, #imm8



Assembler Syntax

Format: ADD Rd, SP | PC, #imm8



SP	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

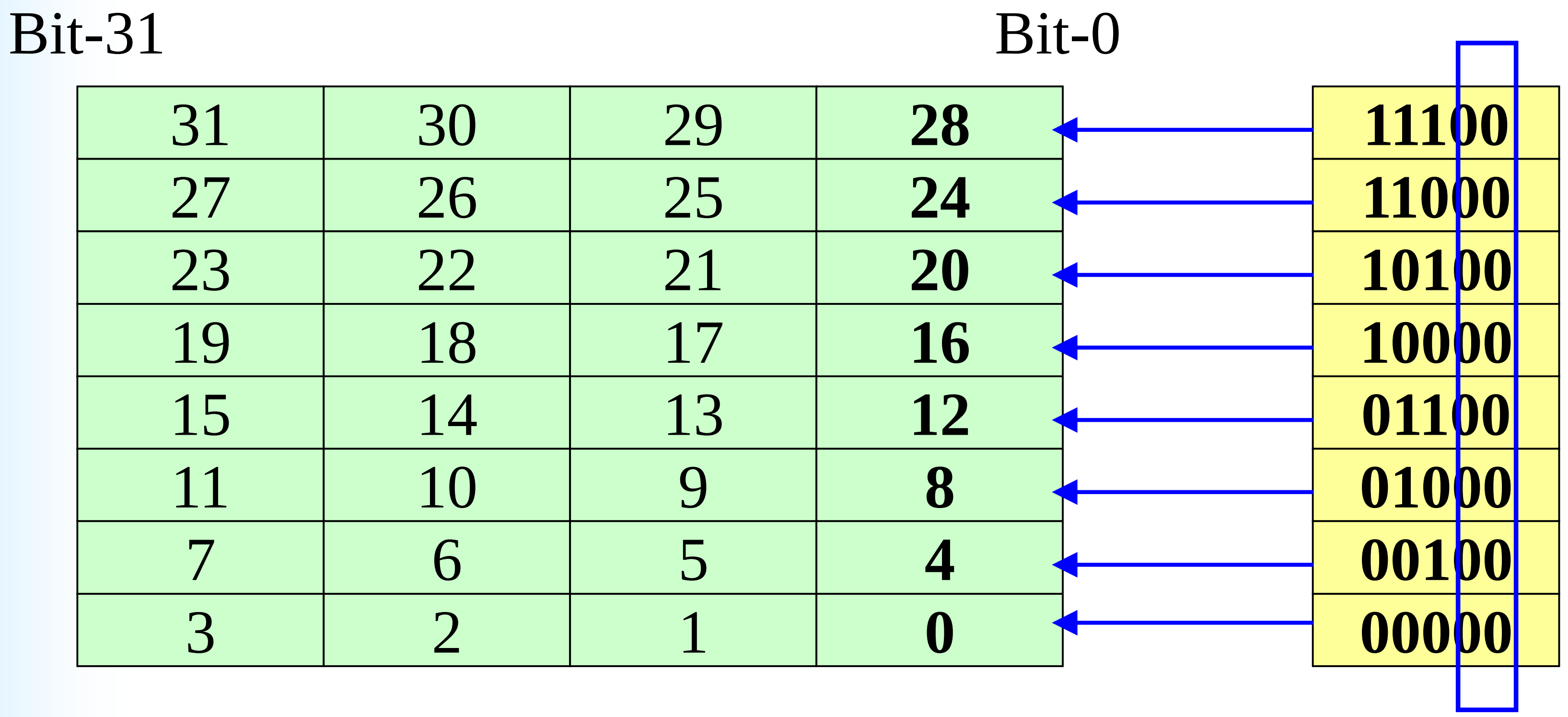
Table 5-13: Load address

Note The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

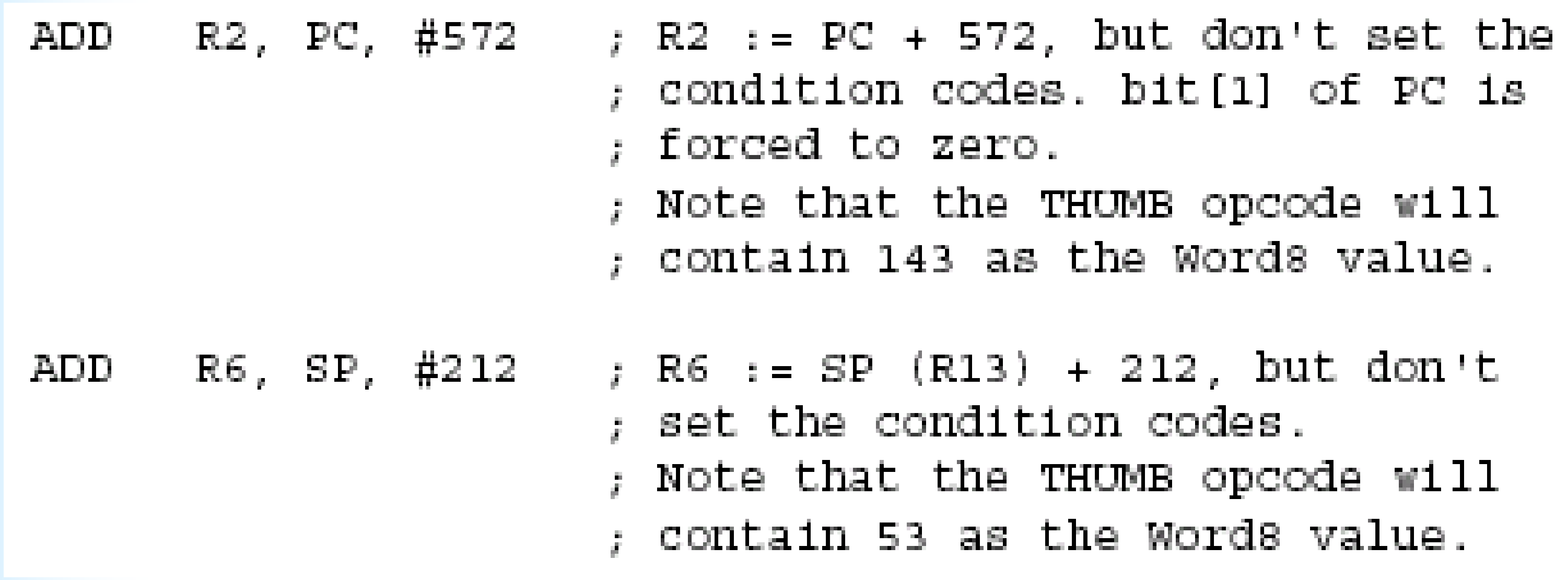
The CPSR condition codes are unaffected by these instructions.

Program Counter (r15)



Bit-31			Bit-0	
31	30	29	**28**	**11100**
27	26	25	**24**	**11000**
23	22	21	**20**	**10100**
19	18	17	**16**	**10000**
15	14	13	**12**	**01100**
11	10	9	**8**	**01000**
7	6	5	**4**	**00100**
3	2	1	**0**	**00000**

Example

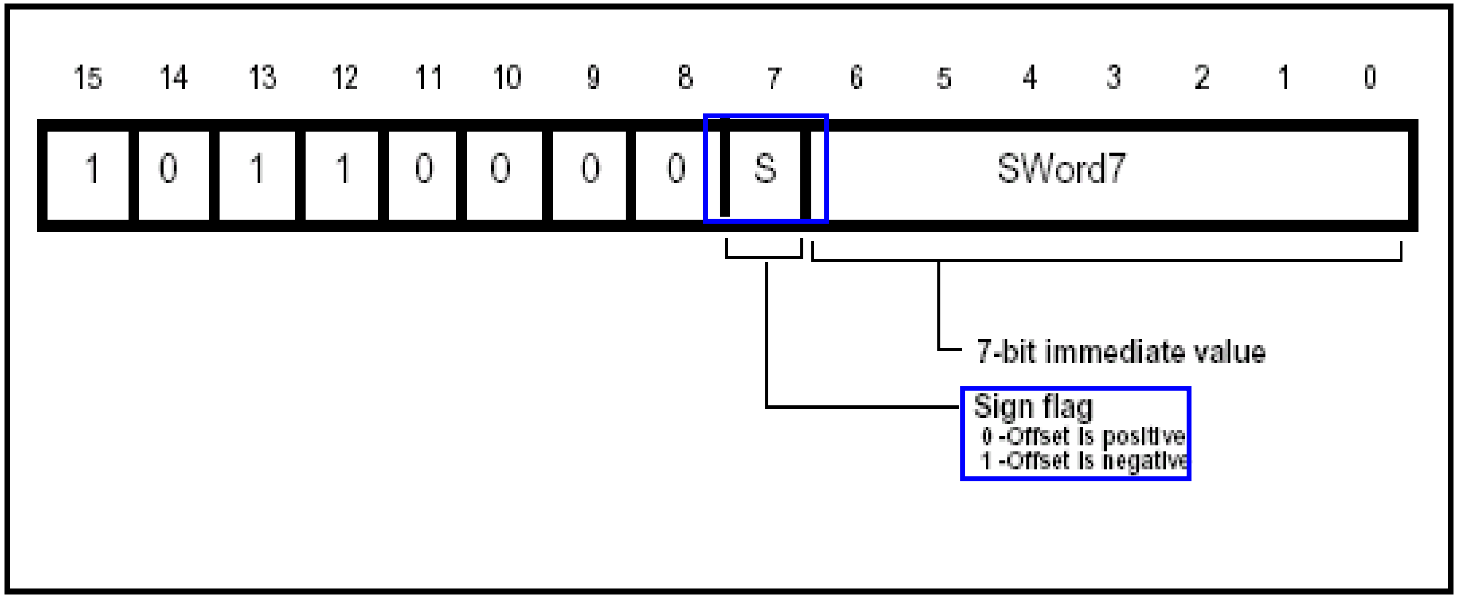


```
ADD    R2, PC, #572    ; R2 := PC + 572, but don't set the
                       ; condition codes. bit[1] of PC is
                       ; forced to zero.
                       ; Note that the THUMB opcode will
                       ; contain 143 as the Word8 value.

ADD    R6, SP, #212    ; R6 := SP (R13) + 212, but don't
                       ; set the condition codes.
                       ; Note that the THUMB opcode will
                       ; contain 53 as the Word8 value.
```

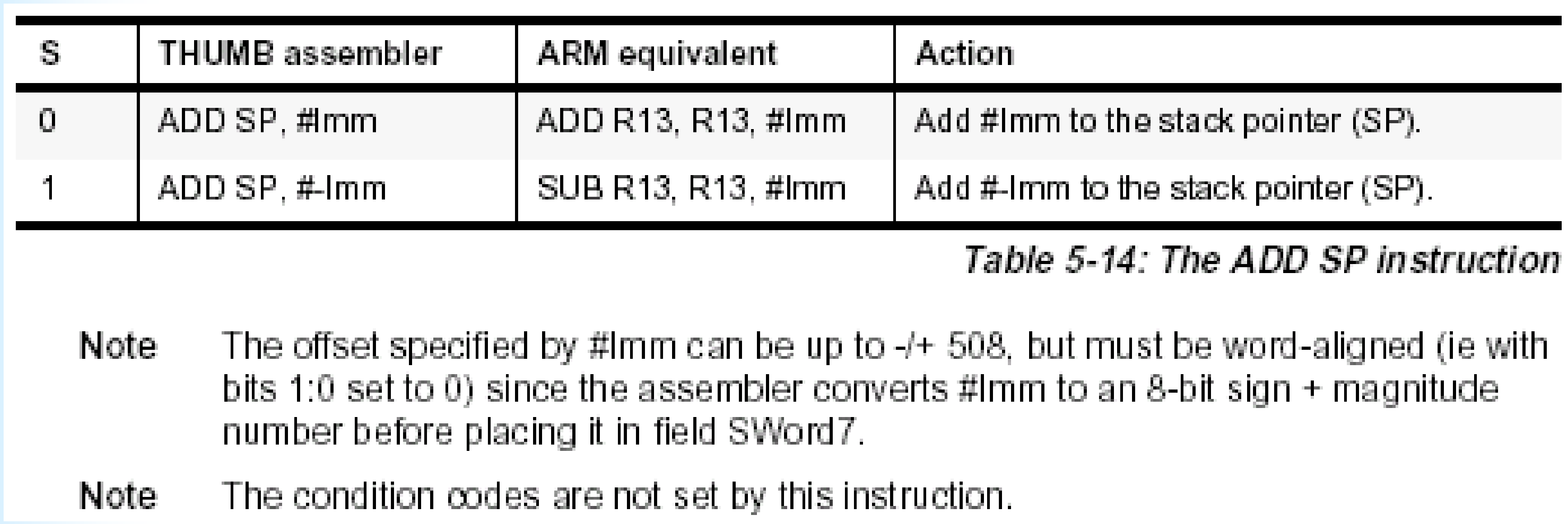
Thumb Data Processing Instructions

- Move shifted register
- Add/subtract
- Move/compare/add/subtract immediate
- ALU operations
- Add PC/SP
- **Add offset to stack pointer**

Add Offset to Stack Pointer



Assembler Syntax



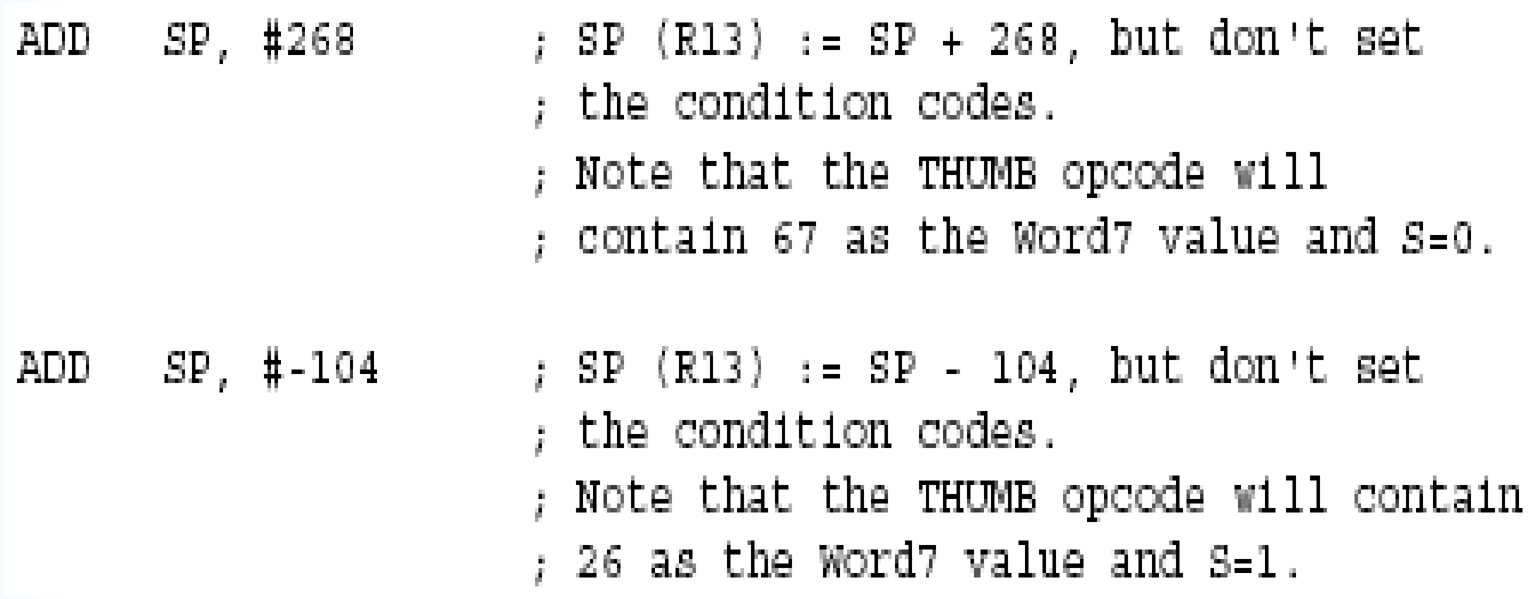
S	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, #-Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

Table 5-14: The ADD SP instruction

Note The offset specified by #Imm can be up to -/+ 508, but must be word-aligned (ie with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7.

Note The condition codes are not set by this instruction.

Example

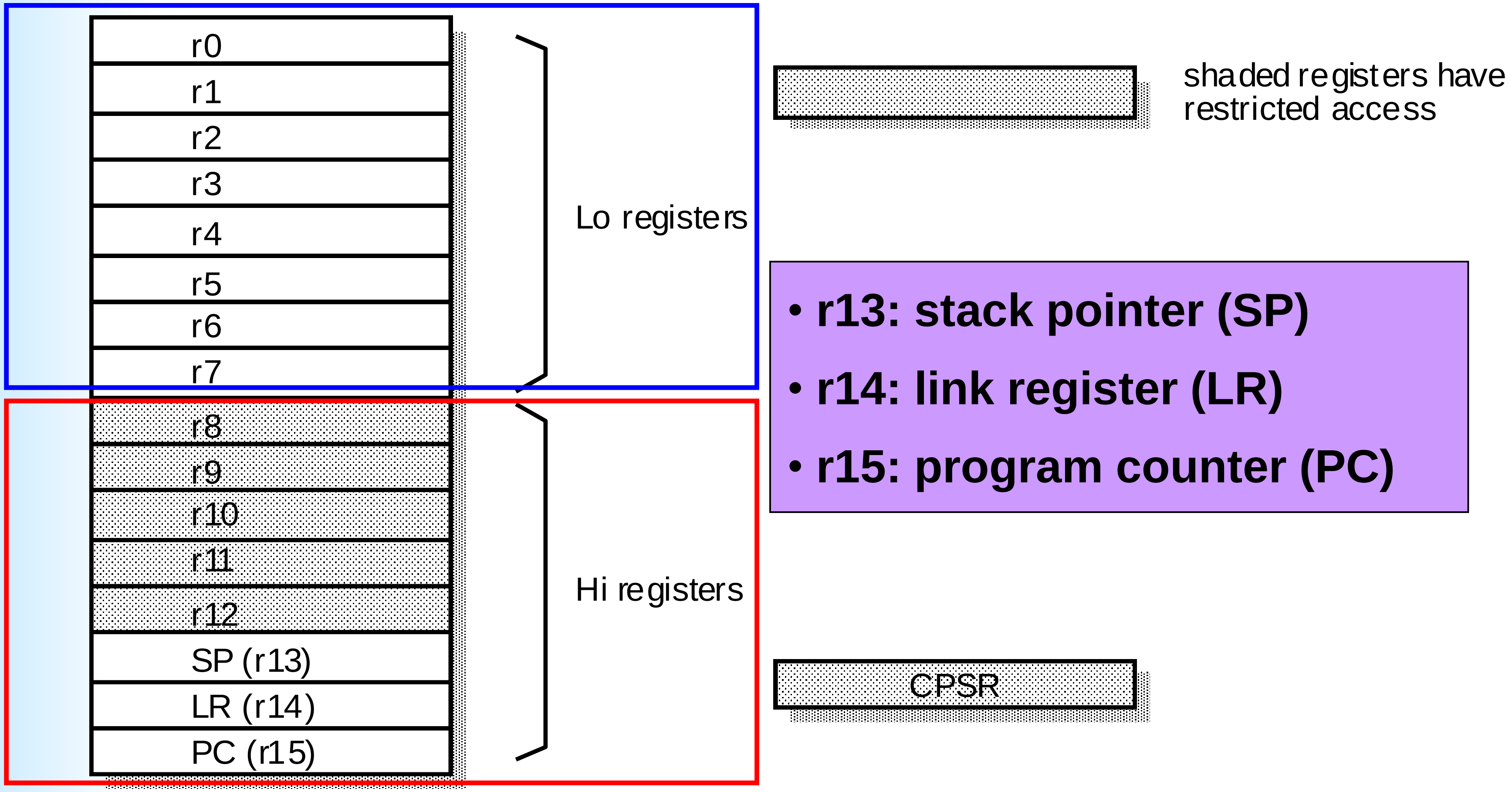


```
ADD    SP, #268          ; SP (R13) := SP + 268, but don't set
                         ; the condition codes.
                         ; Note that the THUMB opcode will
                         ; contain 67 as the Word7 value and S=0.

ADD    SP, #-104         ; SP (R13) := SP - 104, but don't set
                         ; the condition codes.
                         ; Note that the THUMB opcode will contain
                         ; 26 as the Word7 value and S=1.
```

Outline

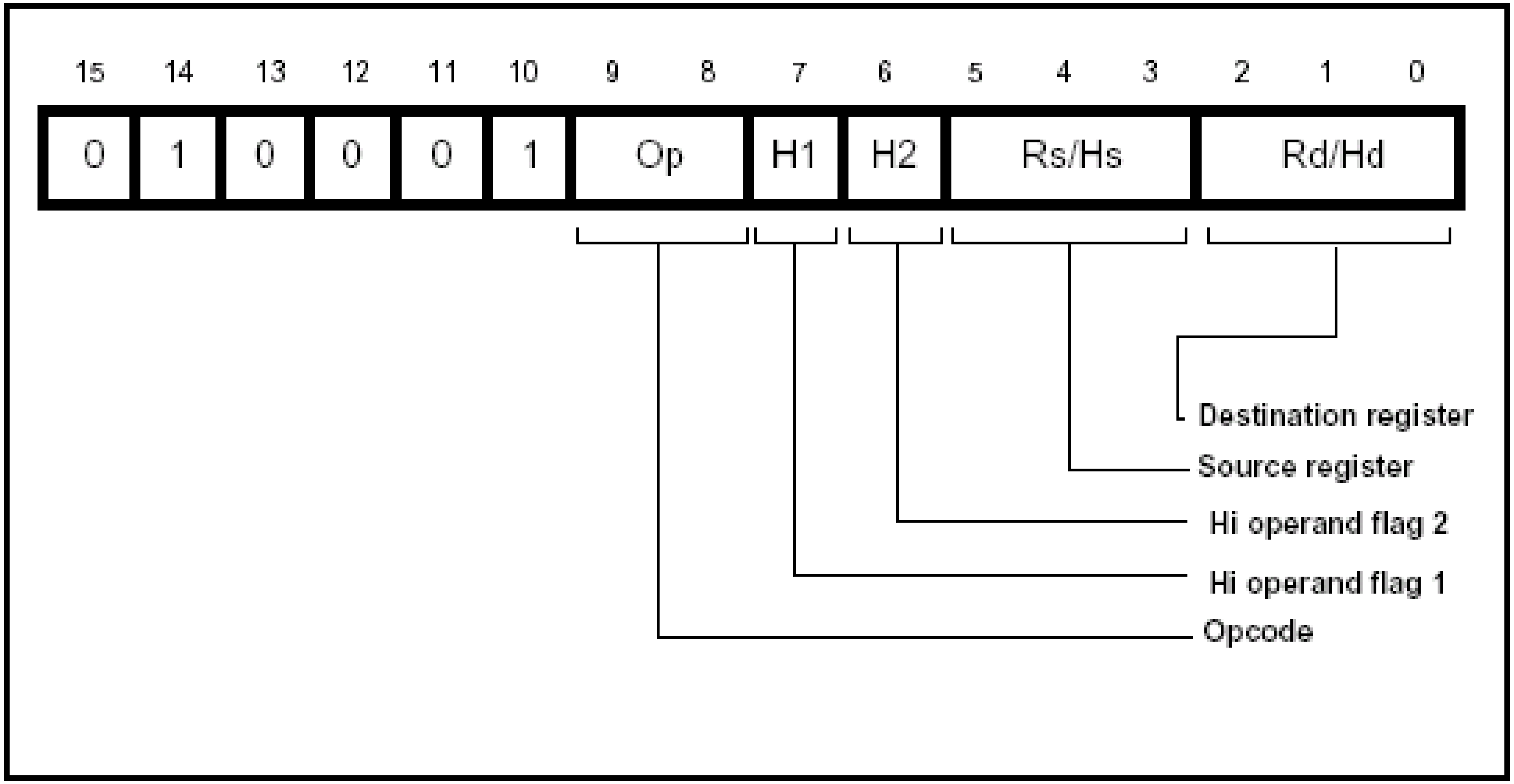
- Introduction
- Branch instruction
- Software interrupt instruction
- Data processing instructions
- **Hi register operations**
- Data transfer instructions

Thumb Accessible Registers



- **r13: stack pointer (SP)**
- **r14: link register (LR)**
- **r15: program counter (PC)**

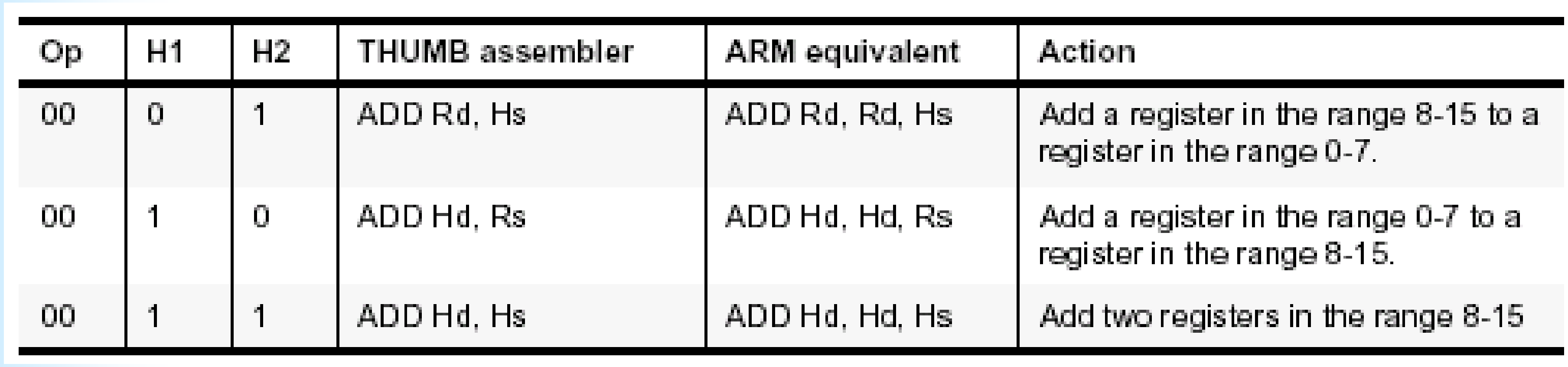
Hi (r8~r15) Register Operations



Assembler Syntax (1)

Instructions that operate with or on the "Hi" register (r8 to r15), in some cases in combination with a "Lo" register

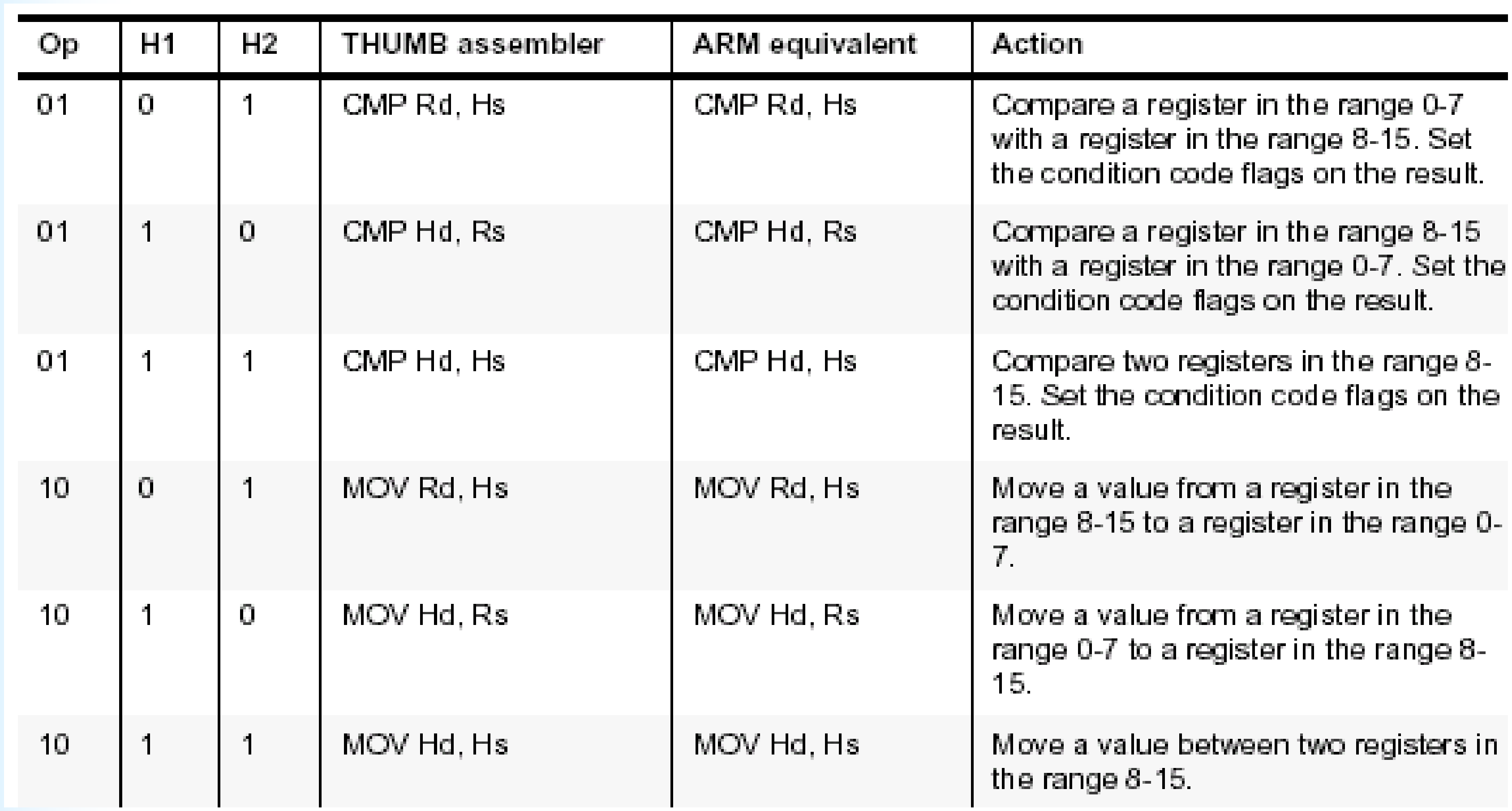
```
<OP>    Rd|Rn, Rm          ; <OP> = ADD | CMP | MOV
```



Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15

Assembler Syntax (2)

The register operands must have one or both "Hi" registers



Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.

Example

Hi register operations

```
      ADD       PC, R5         ; PC := PC + R5 but don't set the
                               ; condition codes.

      CMP       R4, R12        ; Set the condition codes on the
                               ; result of R4 - R12.

      MOV       R15, R14       ; Move R14 (LR) into R15 (PC)
                               ; but don't set the condition codes,
                               ; eg. return from subroutine.
```

Branch and exchange

```
                               ; Switch from THUMB to ARM state.

      ADR       R1,outofTHUMB
                               ; Load address of outofTHUMB
                               ; into R1.
      MOV       R11,R1
      BX        R11            ; Transfer the contents of R11 into
                               ; the PC.
                               ; Bit 0 of R11 determines whether
                               ; ARM or THUMB state is entered, ie.
                               ; ARM state here.
        ...
      ALIGN
      CODE32
outofTHUMB
                               ; Now processing ARM instructions...
```

Outline

- Introduction
- Branch instruction
- Software interrupt instruction
- Data processing instructions
- Hi register operations
- **Data transfer instructions**

Data Transfer Instructions

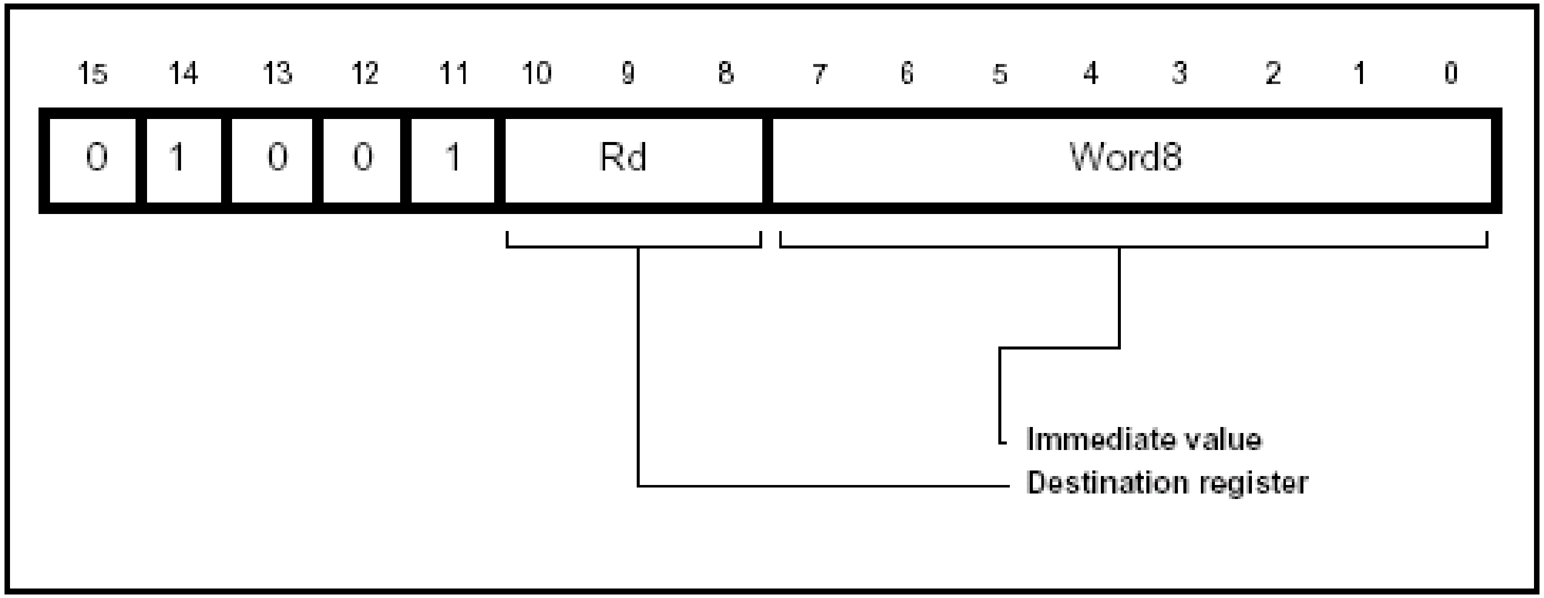
- PC-relative load
- Load/store with register offset
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- Load/store halfword
- SP-relative load/store
- Push/pop registers
- Multiple load/store

Data Transfer Instructions

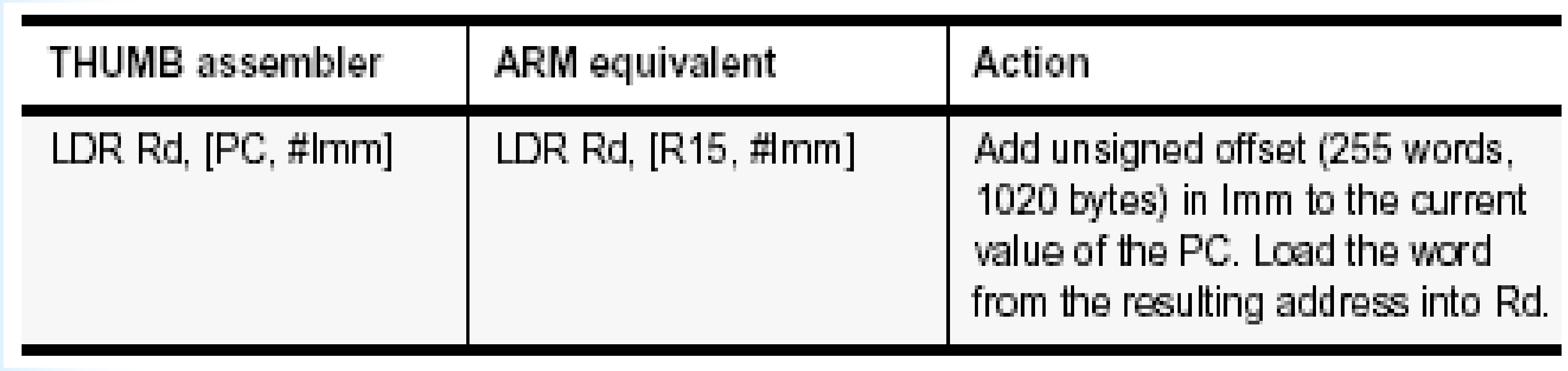
- **PC-relative load**
- Load/store with register offset
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- Load/store halfword
- SP-relative load/store
- Push/pop registers
- Multiple load/store

PC-Relative Load

```
LDR    Rd, [PC, #offset8]
```



Assembler Syntax



THUMB assembler	ARM equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

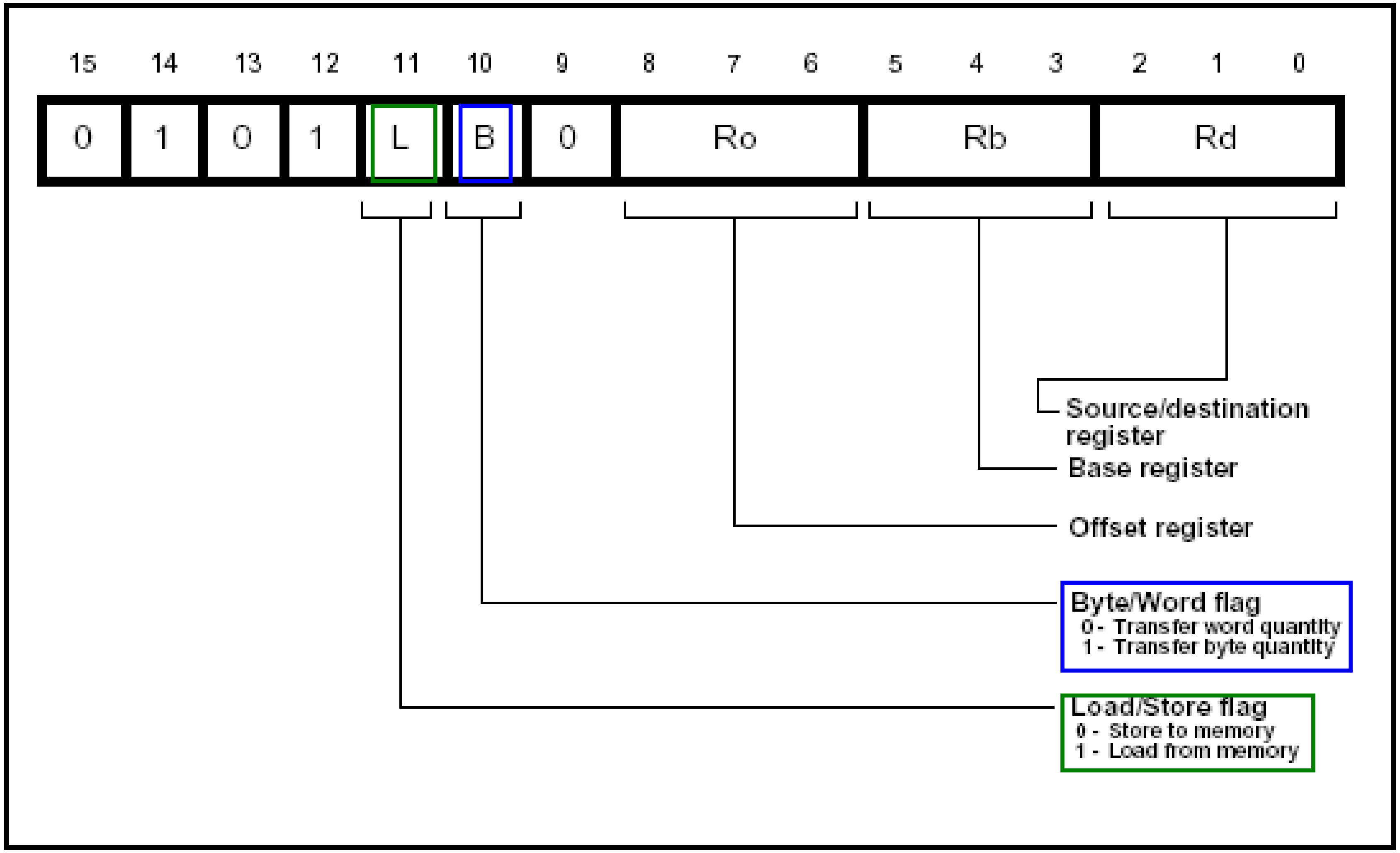
Example

```
LDR R3,[PC,#844]        ; Load into R3 the word found at the
                        ; address formed by adding 844 to PC.
                        ; bit[1] of PC is forced to zero.
                        ; Note that the THUMB opcode will contain
                        ; 211 as the Word8 value.
```

Data Transfer Instructions

- PC-relative load
- **Load/store with register offset**
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- Load/store halfword
- SP-relative load/store
- Push/pop registers
- Multiple load/store

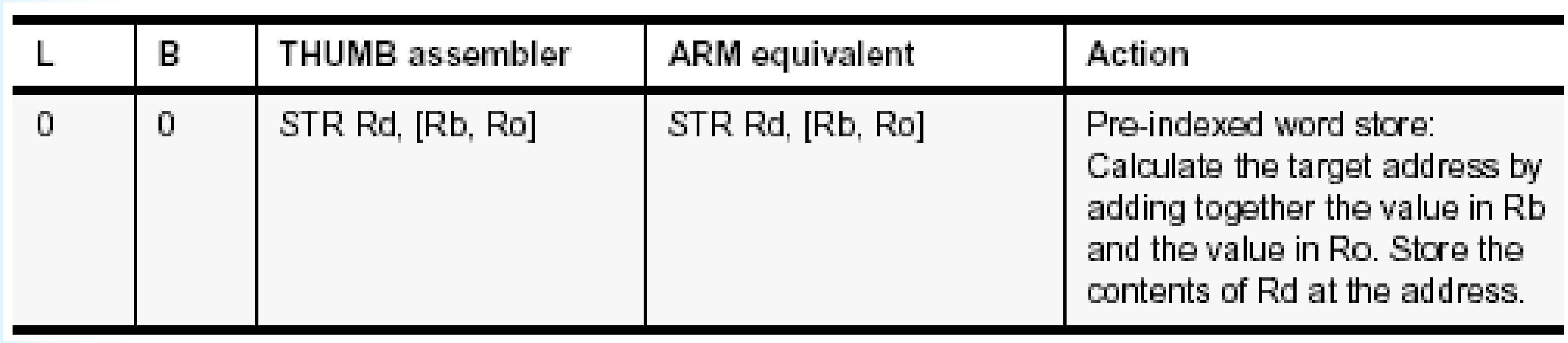
Load/Store with Register Offset



Assembler Syntax (1)

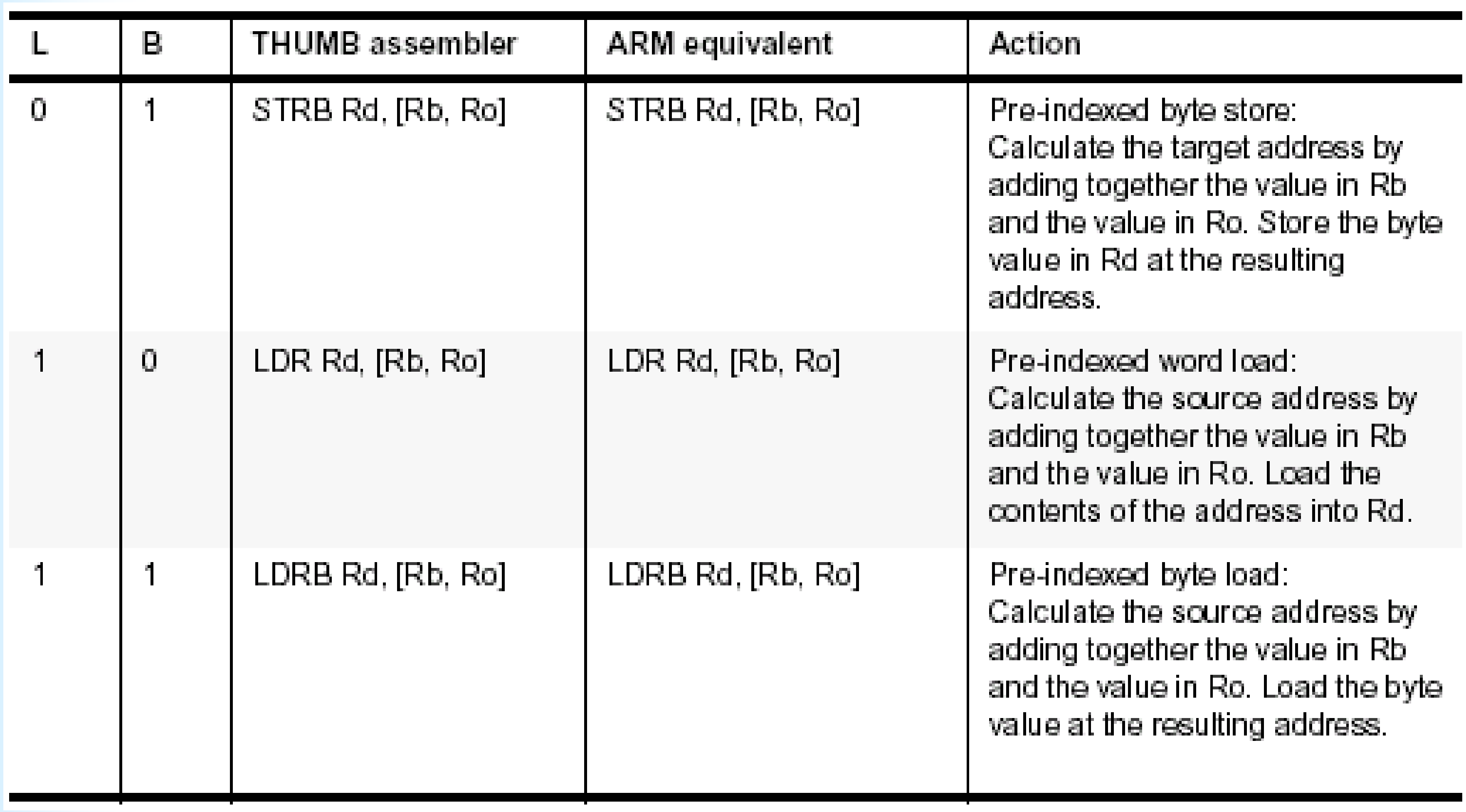
```
LDR|STR{B}    Rd, [Rb, Ro]
```

Byte / Word



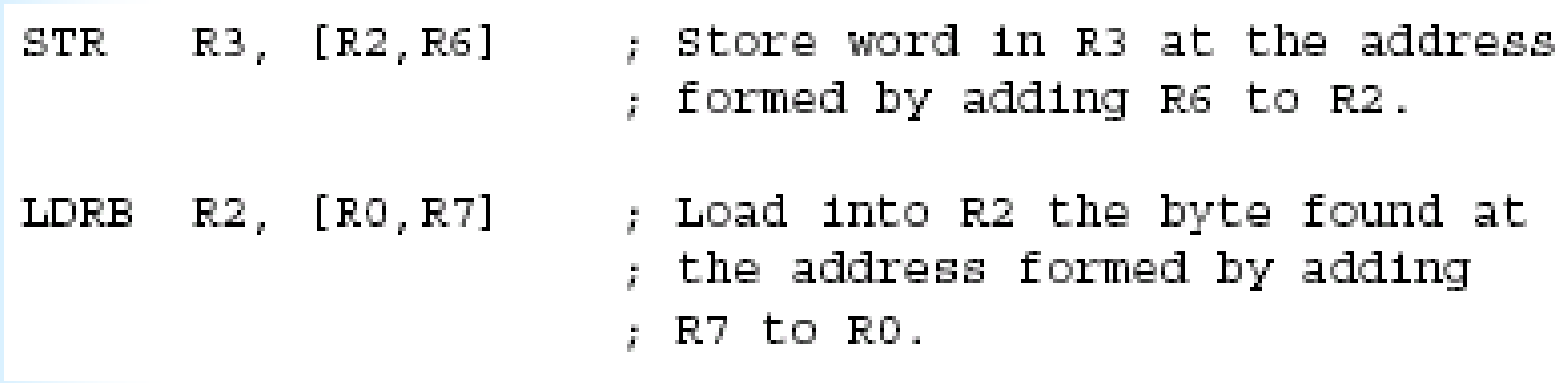
L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.

Assembler Syntax (2)



L	B	THUMB assembler	ARM equivalent	Action
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

Example

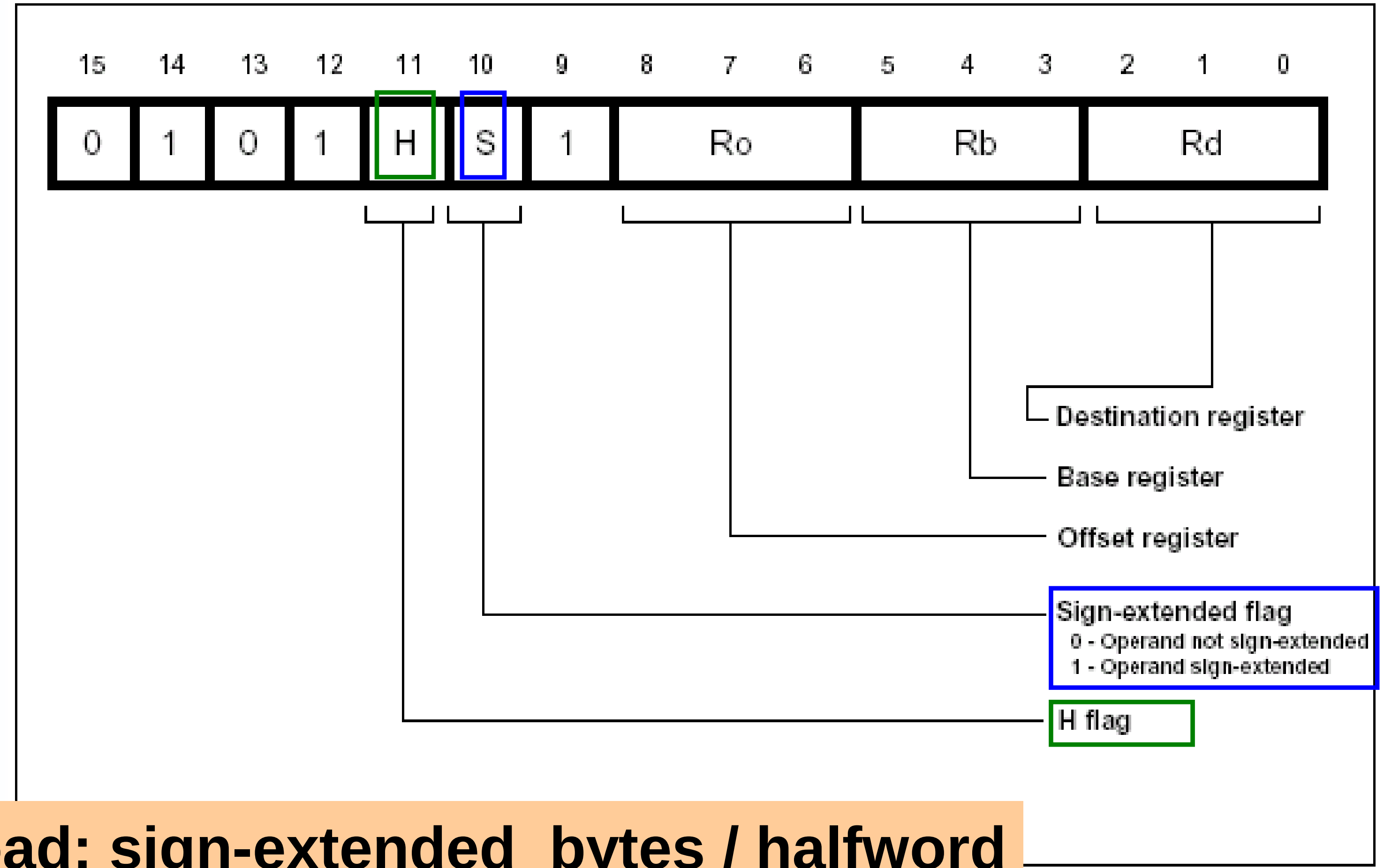


```
STR    R3, [R2,R6]       ; Store word in R3 at the address
                         ; formed by adding R6 to R2.

LDRB   R2, [R0,R7]       ; Load into R2 the byte found at
                         ; the address formed by adding
                         ; R7 to R0.
```

Data Transfer Instructions

- PC-relative load
- Load/store with register offset
- **Load/store sign-extended byte/halfword**
- Load/store with immediate offset
- Load/store halfword
- SP-relative load/store
- Push/pop registers
- Multiple load/store

Load/Store Sign-Extended Byte/Halfword



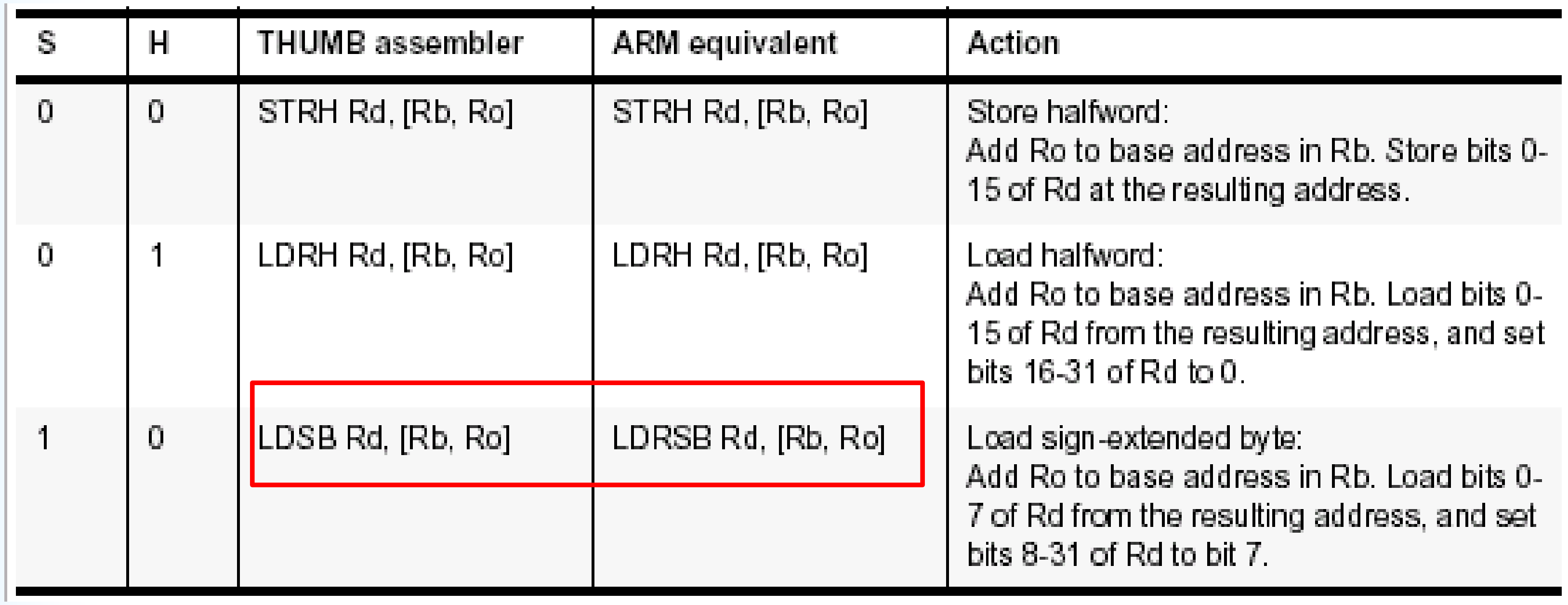
- Load: sign-extended bytes / halfword
- Store: halfword

Assembler Syntax (1)

Sign

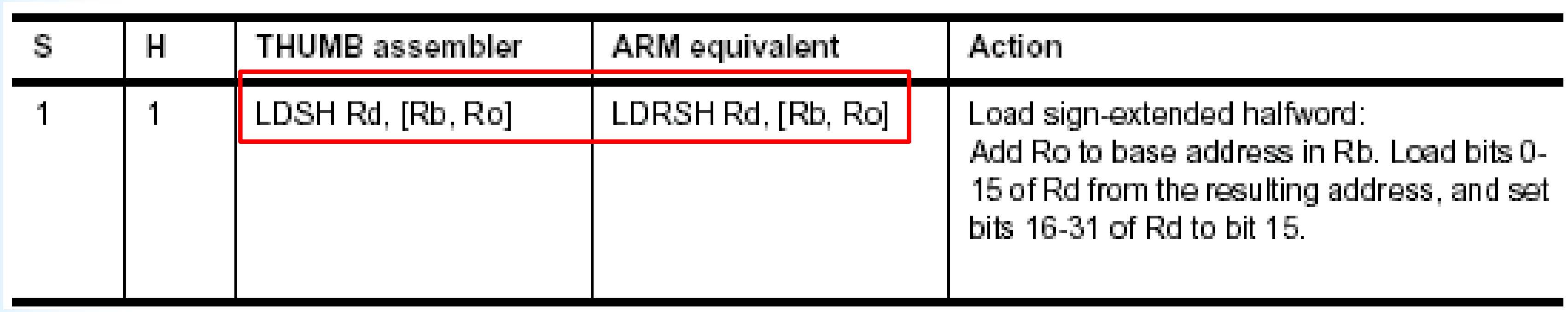
Byte or Halfword

```
LDR{S}{B|H}    Rd, [Rb, Ro]
STR{H}         Rd, [Rb, Ro]
```



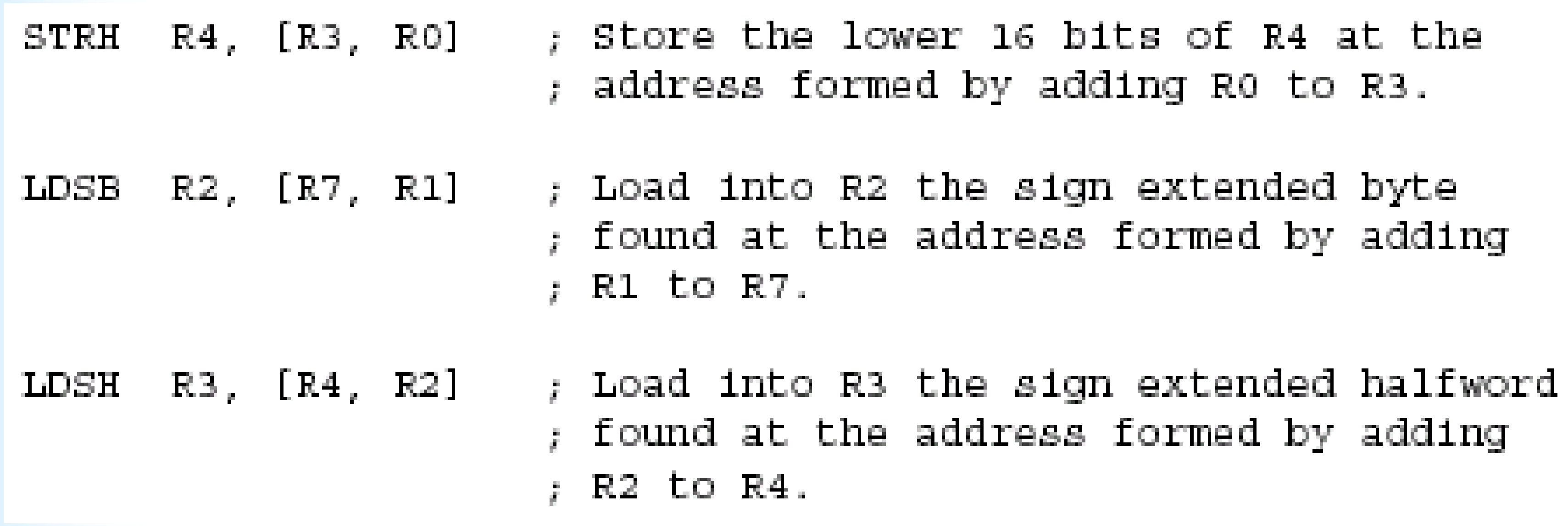
S	H	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.

Assembler Syntax (2)



S	H	THUMB assembler	ARM equivalent	Action
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

Example

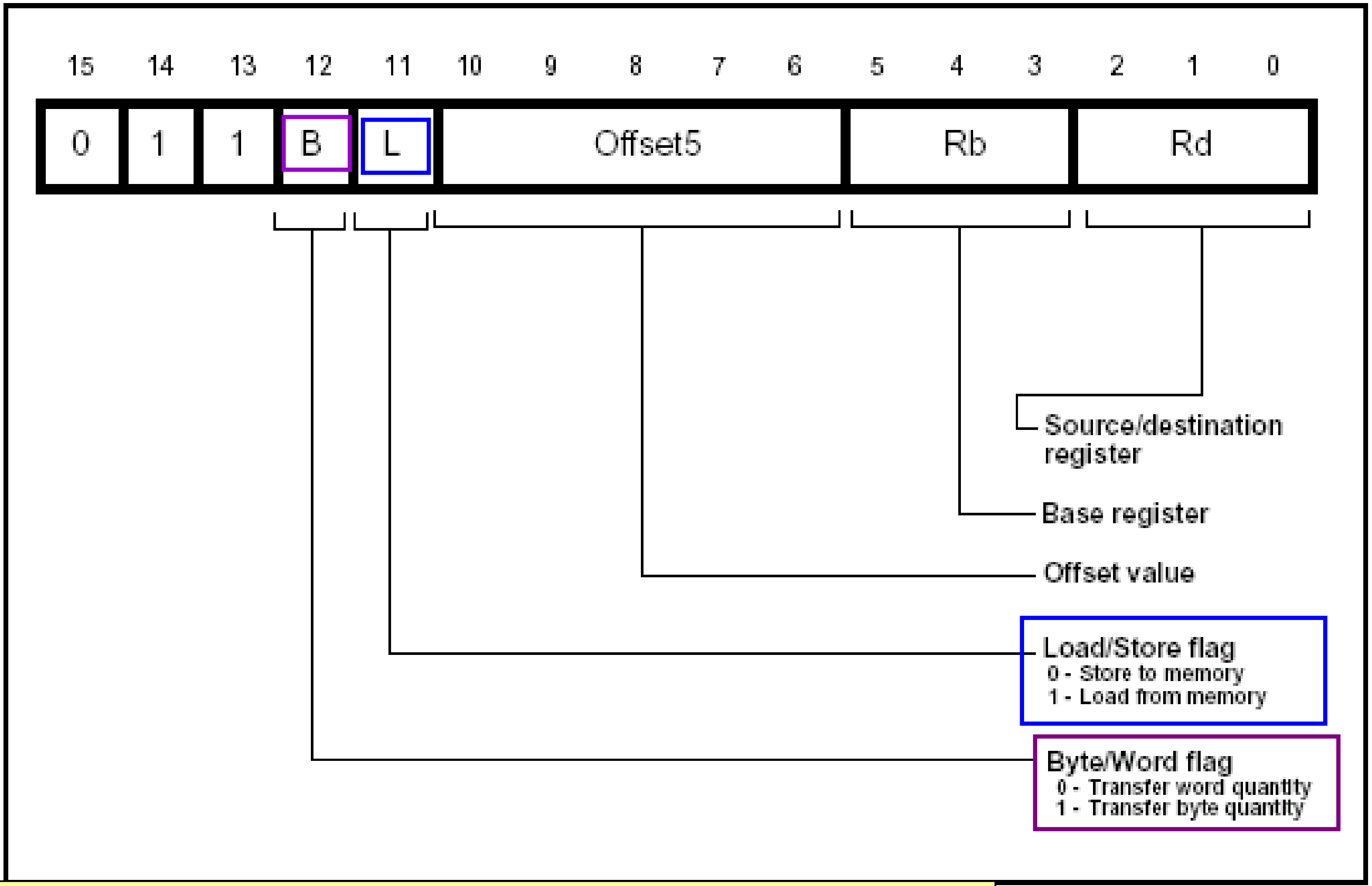


```
STRH  R4, [R3, R0]    ; Store the lower 16 bits of R4 at the
                      ; address formed by adding R0 to R3.

LDSB  R2, [R7, R1]    ; Load into R2 the sign extended byte
                      ; found at the address formed by adding
                      ; R1 to R7.

LDSH  R3, [R4, R2]    ; Load into R3 the sign extended halfword
                      ; found at the address formed by adding
                      ; R2 to R4.
```

Data Transfer Instructions

- PC-relative load
- Load/store with register offset
- Load/store sign-extended byte/halfword
- **Load/store with immediate offset**
- Load/store halfword
- SP-relative load/store
- Push/pop registers
- Multiple load/store

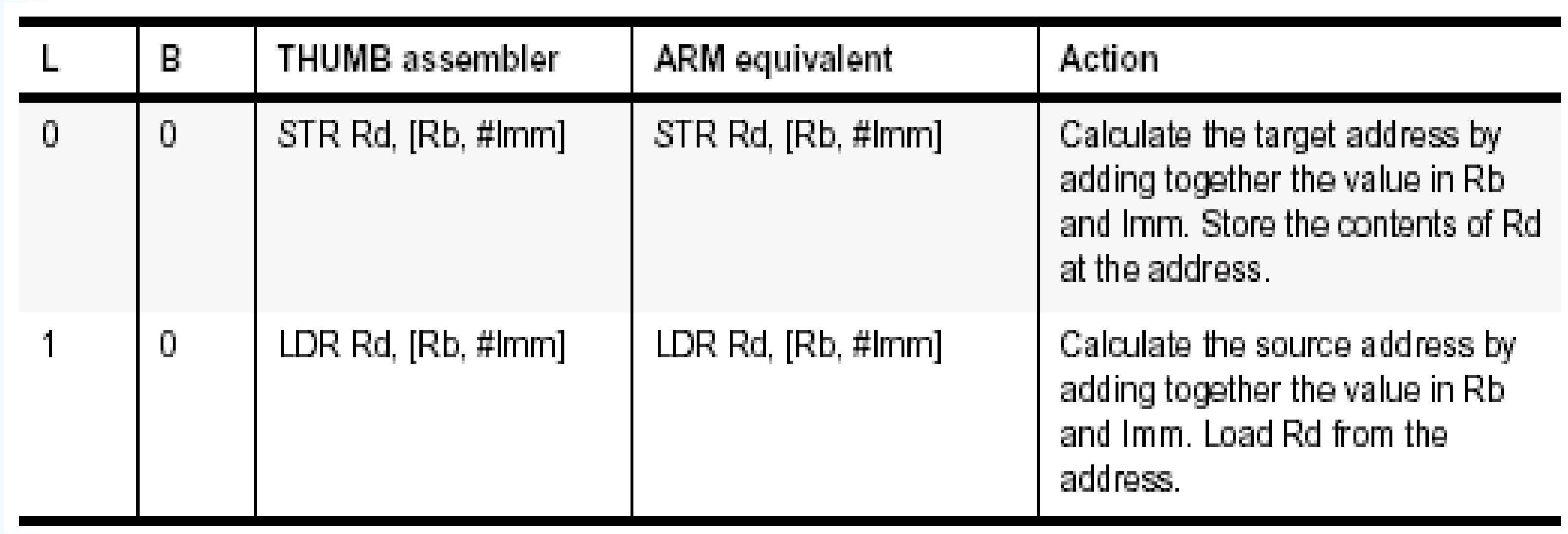
Load/Store with Immediate Offset



```
LDR|STR{B}   Rd, [Rb, #offset5]
```

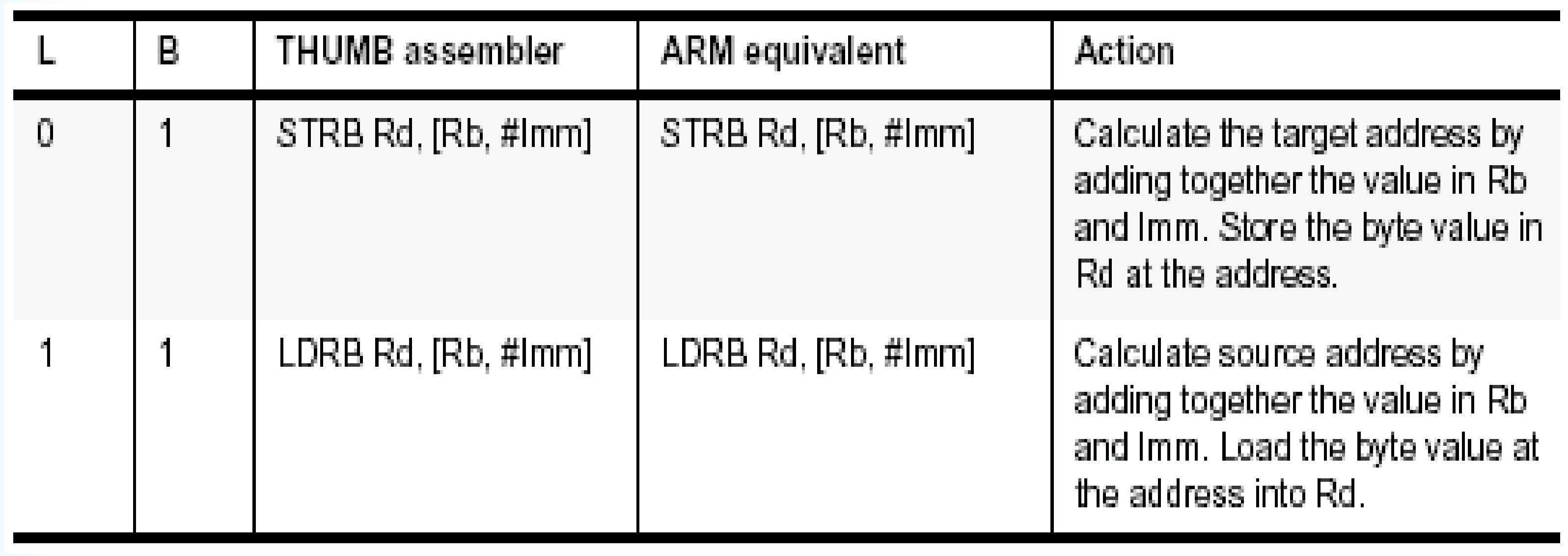
Assembler Syntax (1)

```
LDR|STR{B}   Rd, [Rn, #offset5]
```



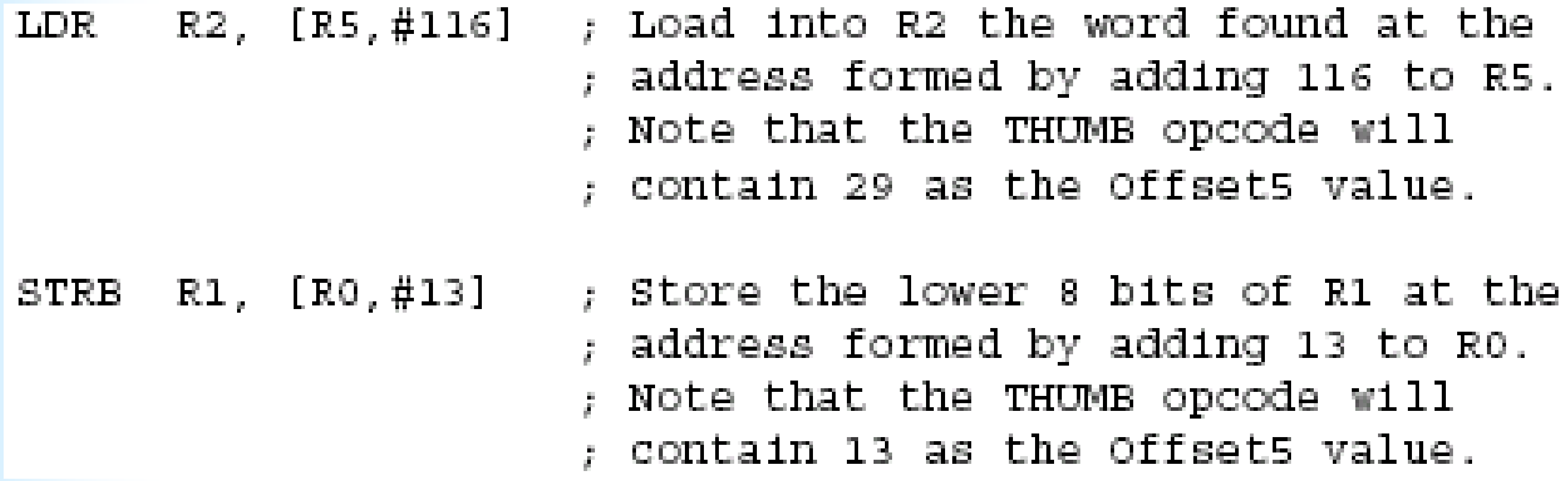
L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.

Assembler Syntax (2)



L	B	THUMB assembler	ARM equivalent	Action
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

Example



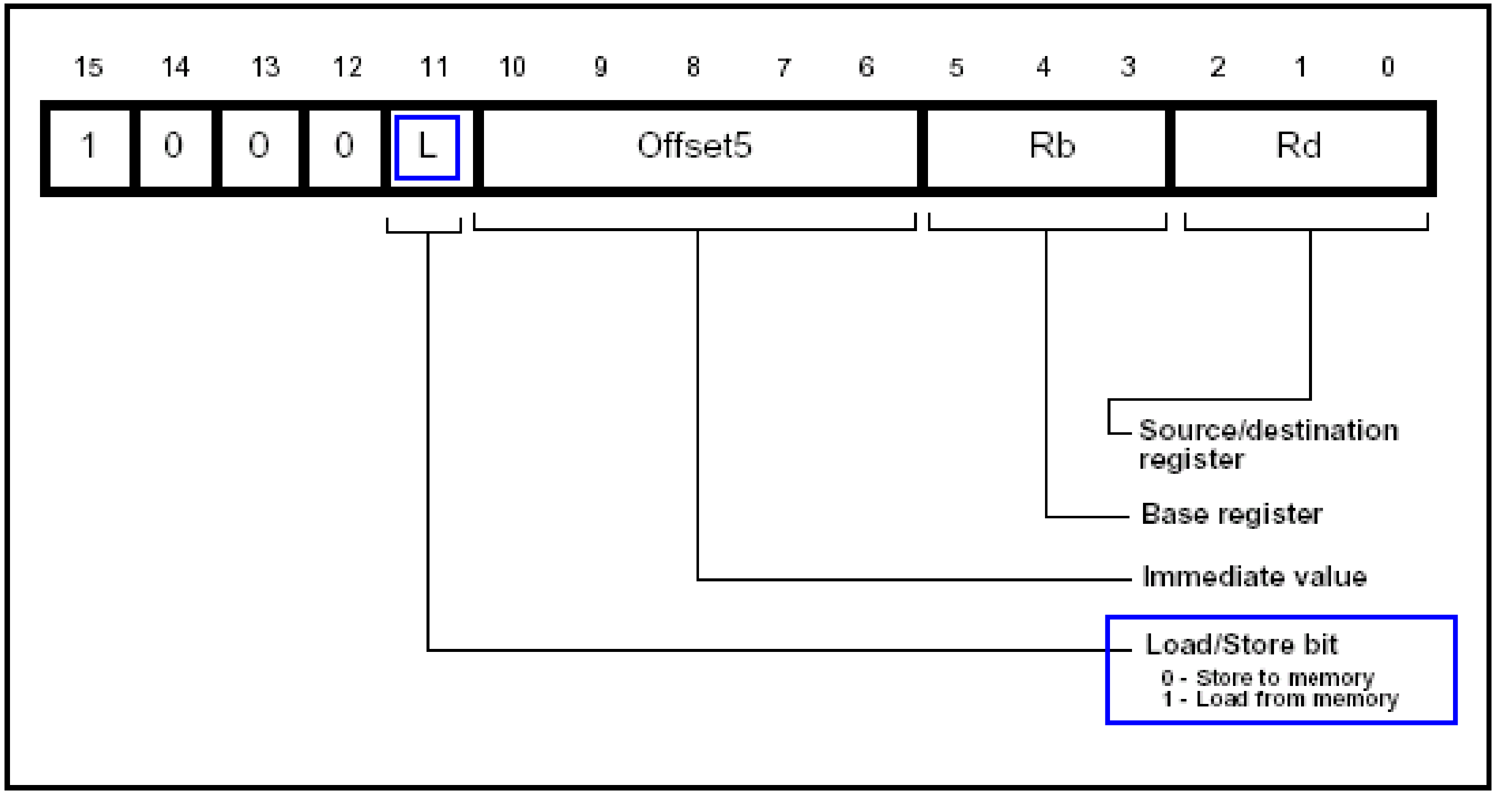
```
LDR    R2, [R5,#116]    ; Load into R2 the word found at the
                        ; address formed by adding 116 to R5.
                        ; Note that the THUMB opcode will
                        ; contain 29 as the Offset5 value.

STRB   R1, [R0,#13]     ; Store the lower 8 bits of R1 at the
                        ; address formed by adding 13 to R0.
                        ; Note that the THUMB opcode will
                        ; contain 13 as the Offset5 value.
```

Data Transfer Instructions

- PC-relative load
- Load/store with register offset
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- **Load/store halfword**
- SP-relative load/store
- Push/pop registers
- Multiple load/store

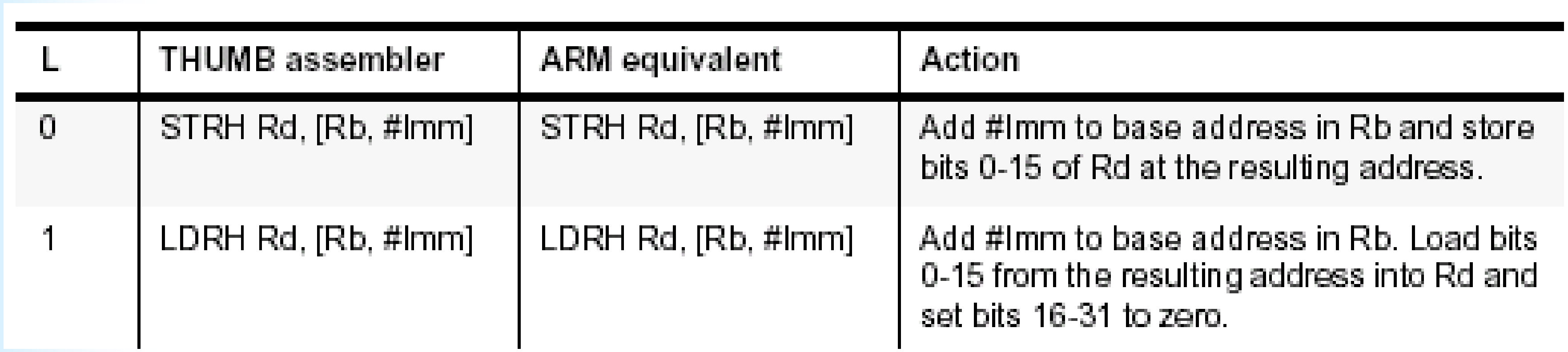
Load/Store Halfword

```
LDRH|STRH   Rd, [Rn, #offset5]
```



Assembler Syntax

```
LDRH|STRH    Rd, [Rn, #offset5]
```

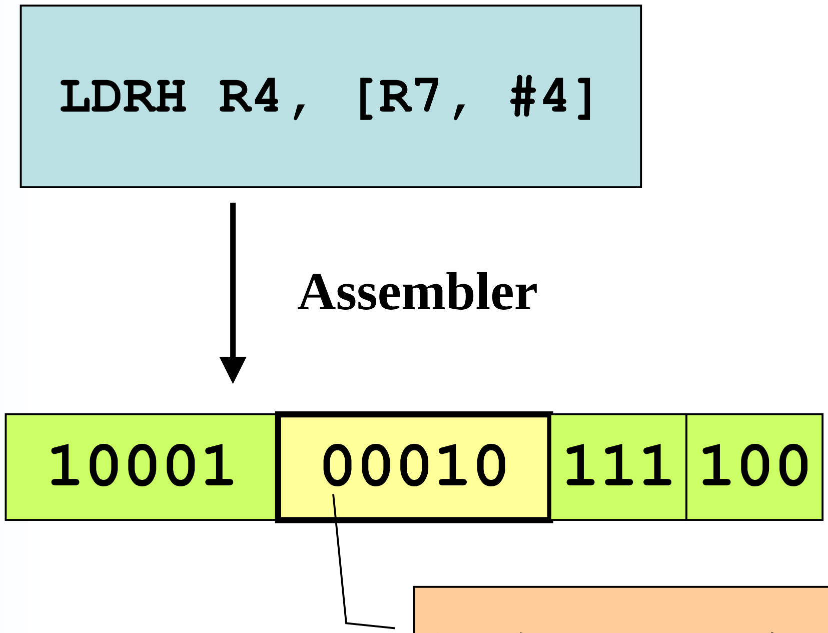


L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0-15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

Example (1)

```
STRH   R6, [R1, #56]    ; Store the lower 16 bits of R6 at
                        ; the address formed by adding 56
                        ; R1.
                        ; Note that the THUMB opcode will
                        ; contain 28 as the Offset5 value.

LDRH   R4, [R7, #4]     ; Load into R4 the halfword found at
                        ; the address formed by adding 4 to R7.
                        ; Note that the THUMB opcode will contain
                        ; 2 as the Offset5 value.
```

Example (2)



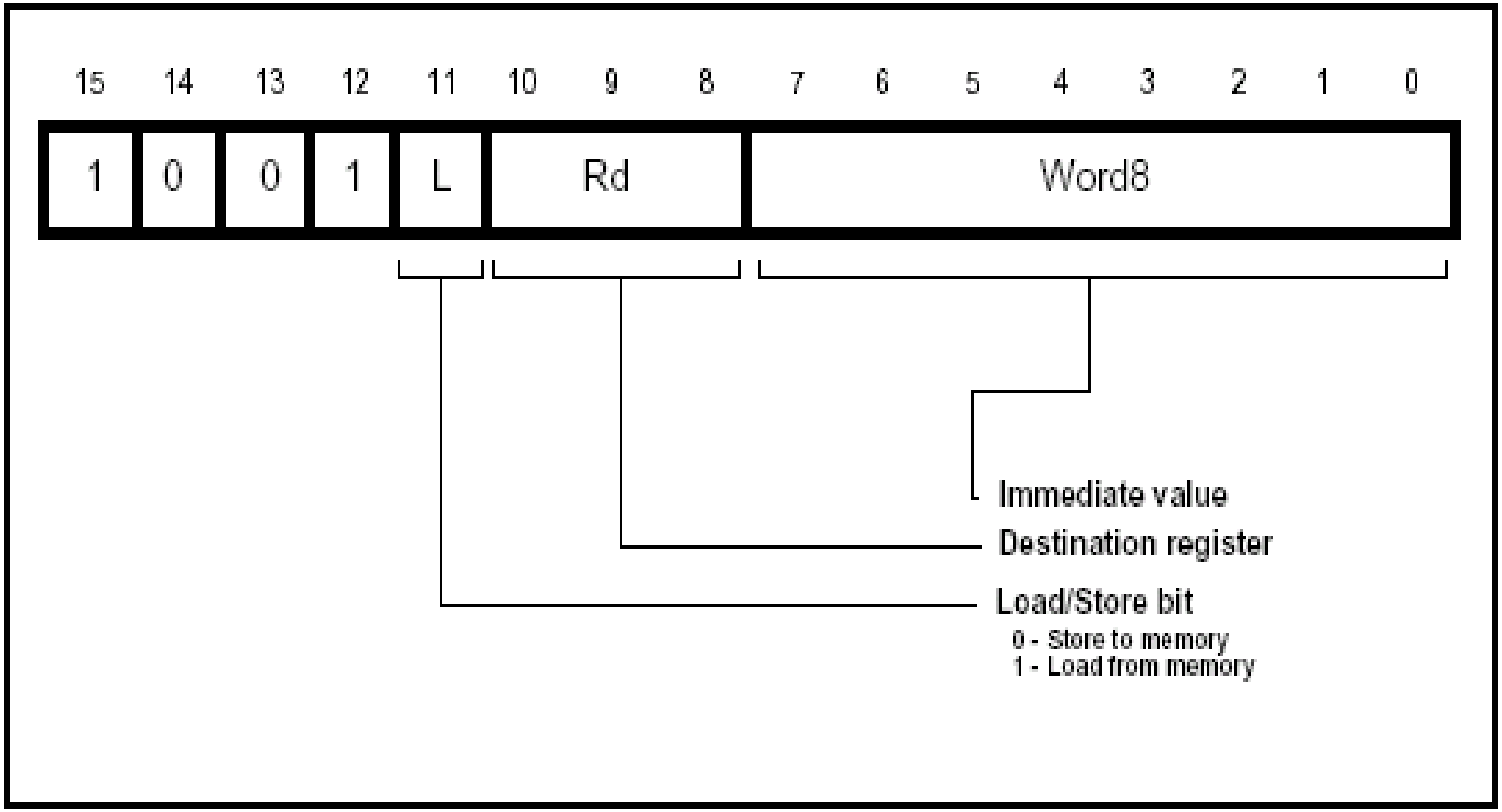
因為load一個half-word，所以processor執行前，會把此欄為的值先乘2

Data Transfer Instructions

- PC-relative load
- Load/store with register offset
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- Load/store halfword
- **SP-relative load/store**
- Push/pop registers
- Multiple load/store

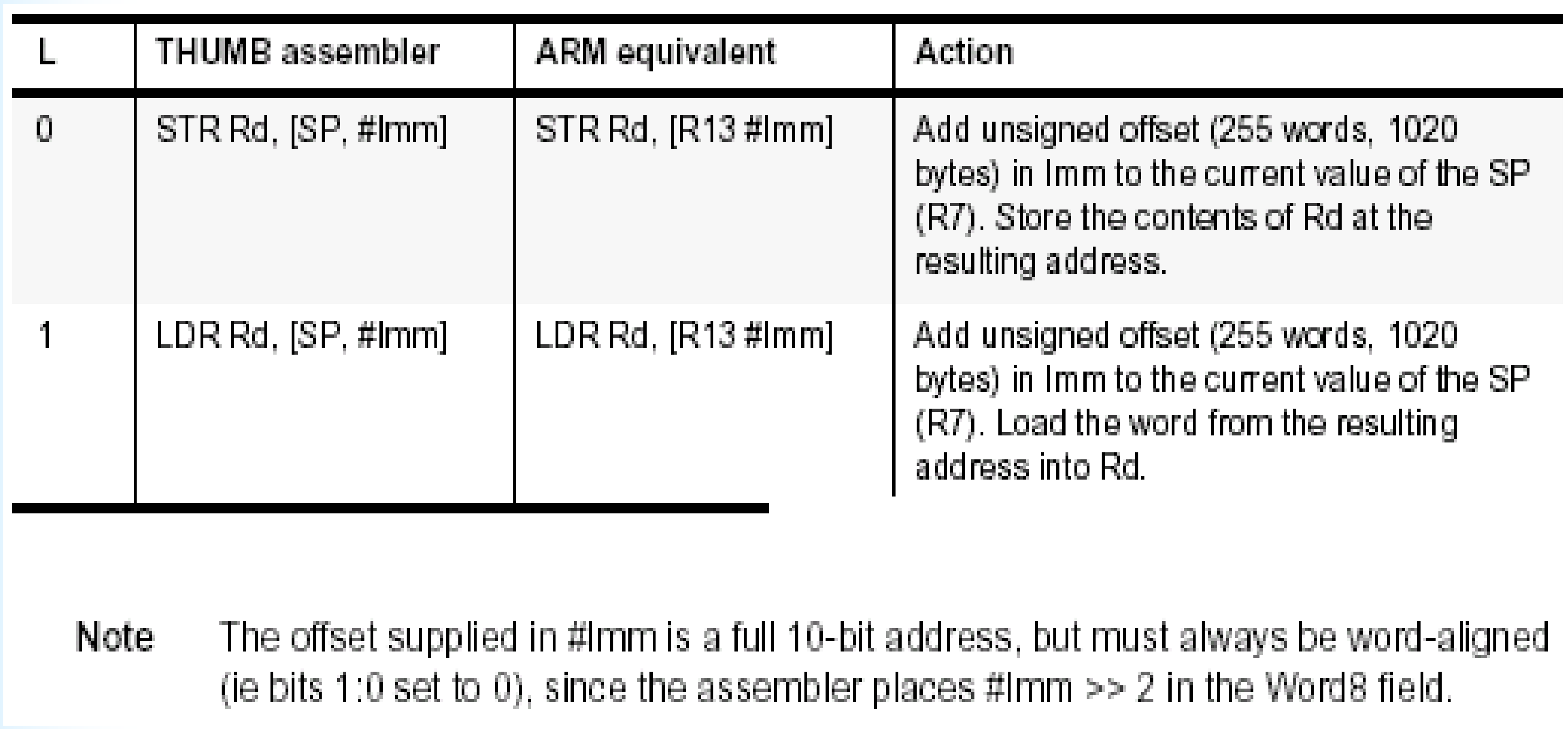
SP-Relative Load/Store

```
LDR|STR   Rd, [SP, #offset8]
```



Assembler Syntax

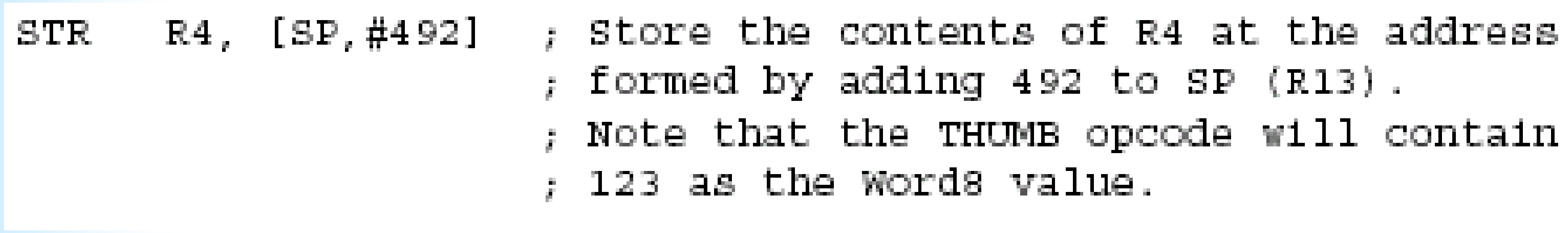
```
LDR|STR   Rd, [SP, #offset8]
```



L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

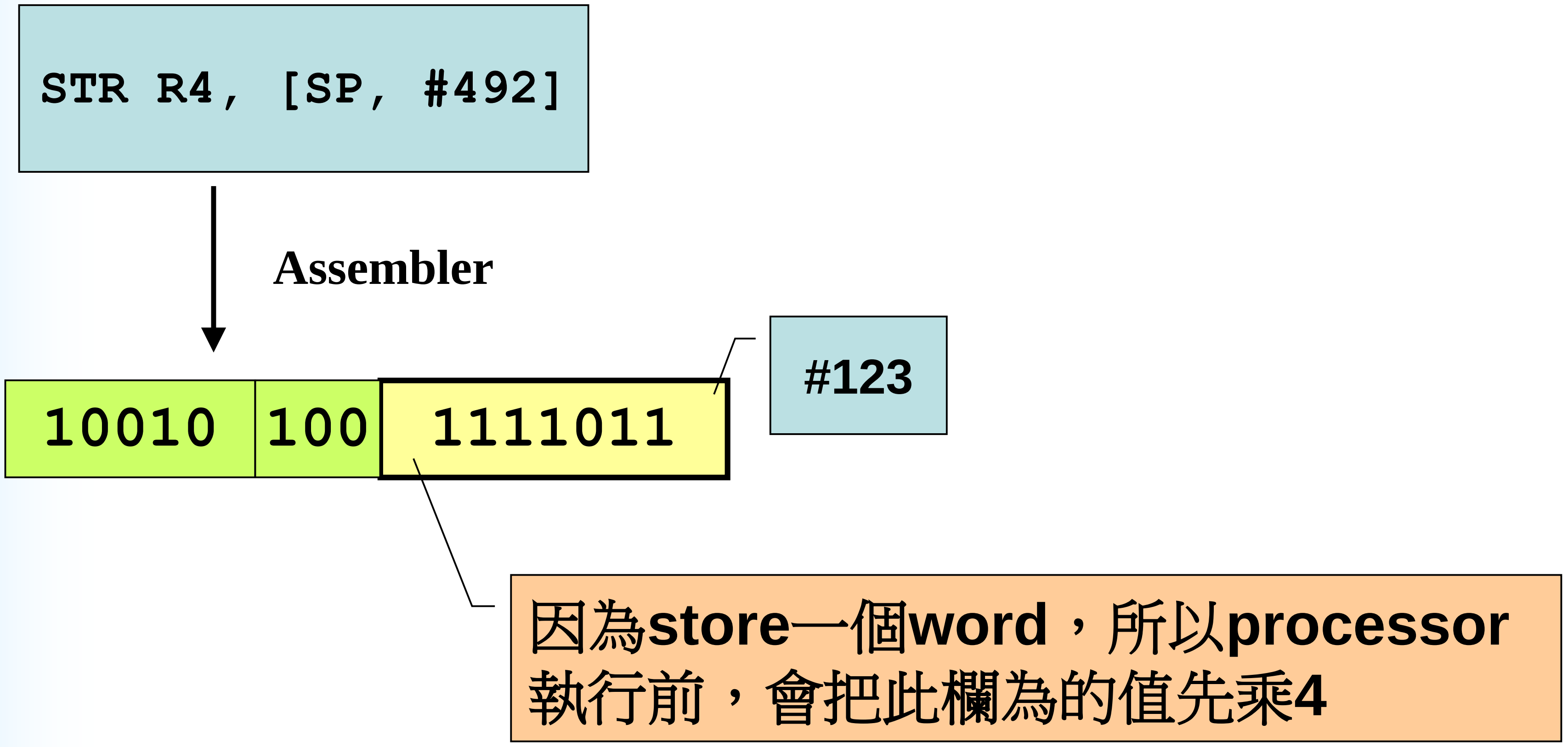
Note The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.

Example (1)



```
STR    R4, [SP,#492]   ; Store the contents of R4 at the address
                       ; formed by adding 492 to SP (R13).
                       ; Note that the THUMB opcode will contain
                       ; 123 as the Word8 value.
```

Example (2)



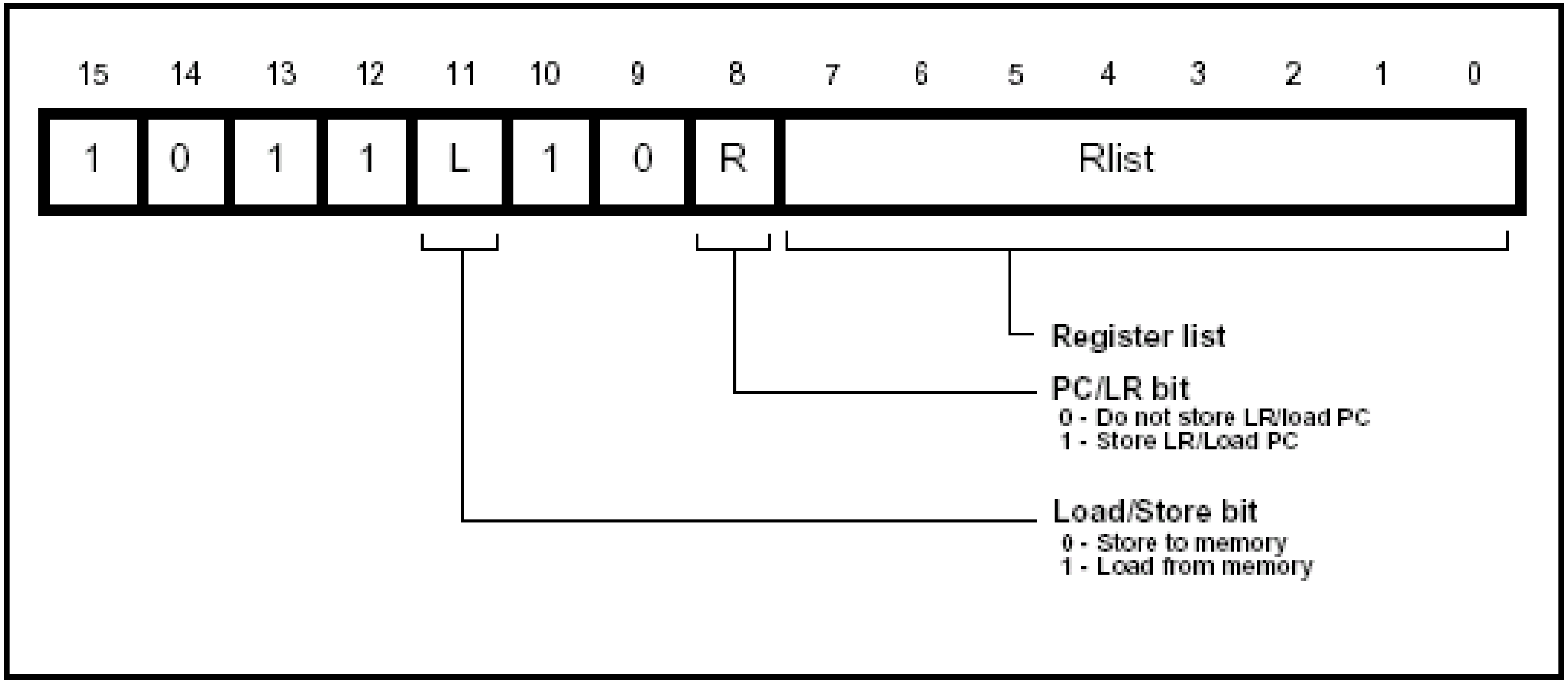
STR R4, [SP, #492]
Assembler
10010
100
1111011
#123
因為store一個word，所以processor執行前，會把此欄為的值先乘4

Data Transfer Instructions

- PC-relative load
- Load/store with register offset
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- Load/store halfword
- SP-relative load/store
- **Push/pop registers**
- Multiple load/store

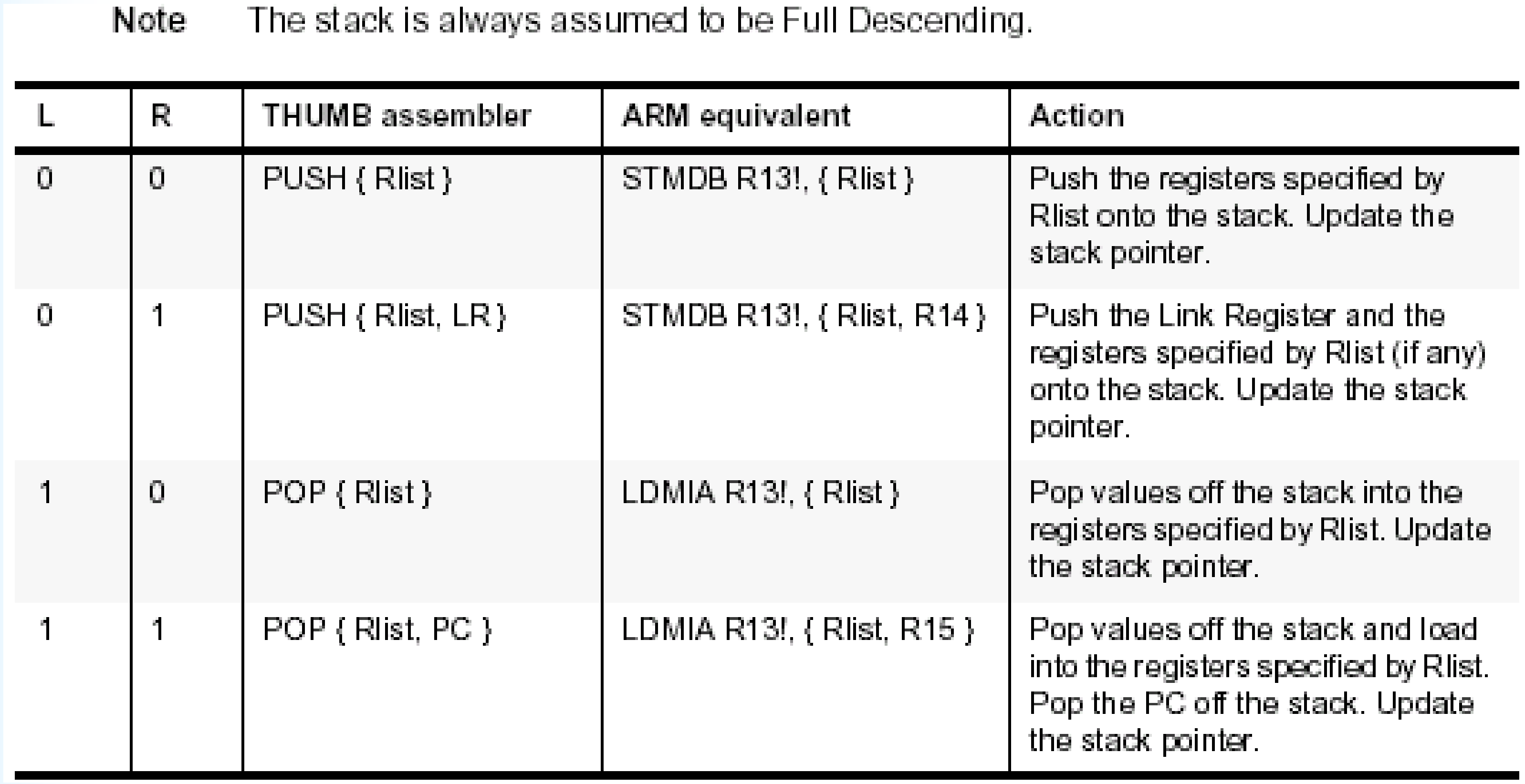
Push/Pop Registers

- `POP    {<reg list> {,PC}}`
- `PUSH   {<reg list> {,LR}}`



Assembler Syntax

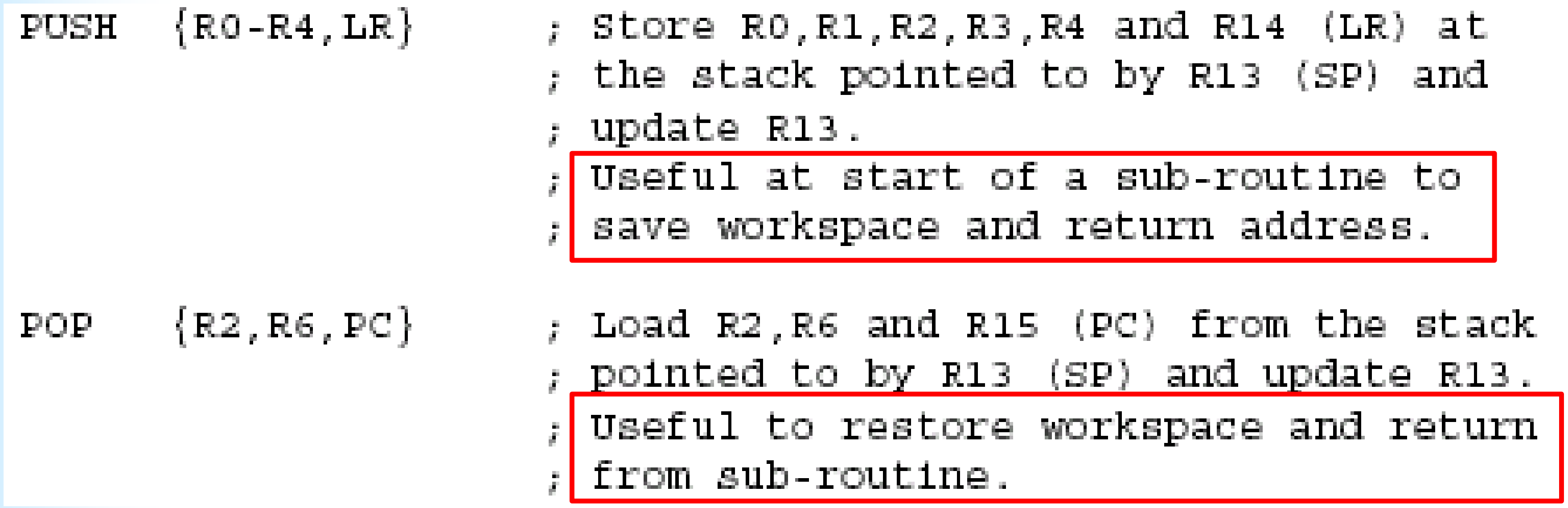
- **POP {<reg list> {,PC}}**
- **PUSH {<reg list> {,LR}}**



Note The stack is always assumed to be Full Descending.

L	R	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, { Rlist, R15 }	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

Example



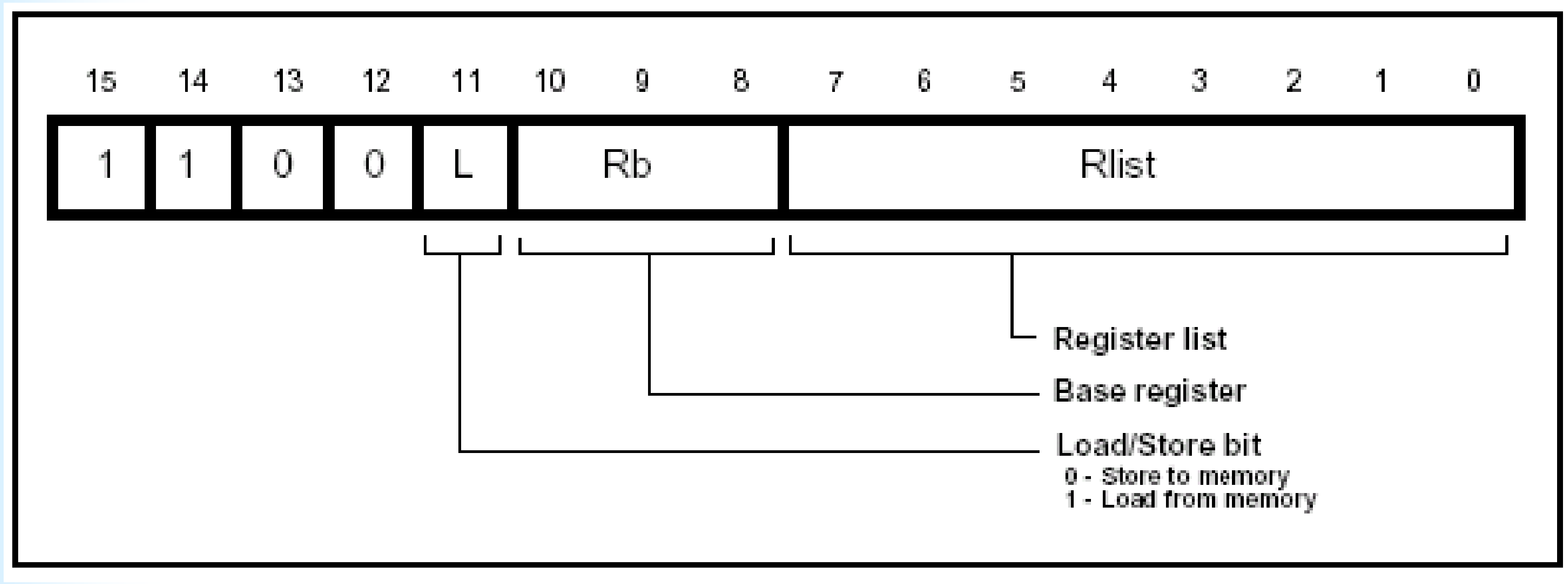
```
PUSH   {R0-R4,LR}          ; Store R0,R1,R2,R3,R4 and R14 (LR) at
                           ; the stack pointed to by R13 (SP) and
                           ; update R13.
                           ; Useful at start of a sub-routine to
                           ; save workspace and return address.

POP    {R2,R6,PC}          ; Load R2,R6 and R15 (PC) from the stack
                           ; pointed to by R13 (SP) and update R13.
                           ; Useful to restore workspace and return
                           ; from sub-routine.
```

Data Transfer Instructions

- PC-relative load
- Load/store with register offset
- Load/store sign-extended byte/halfword
- Load/store with immediate offset
- Load/store halfword
- SP-relative load/store
- Push/pop registers
- **Multiple load/store**

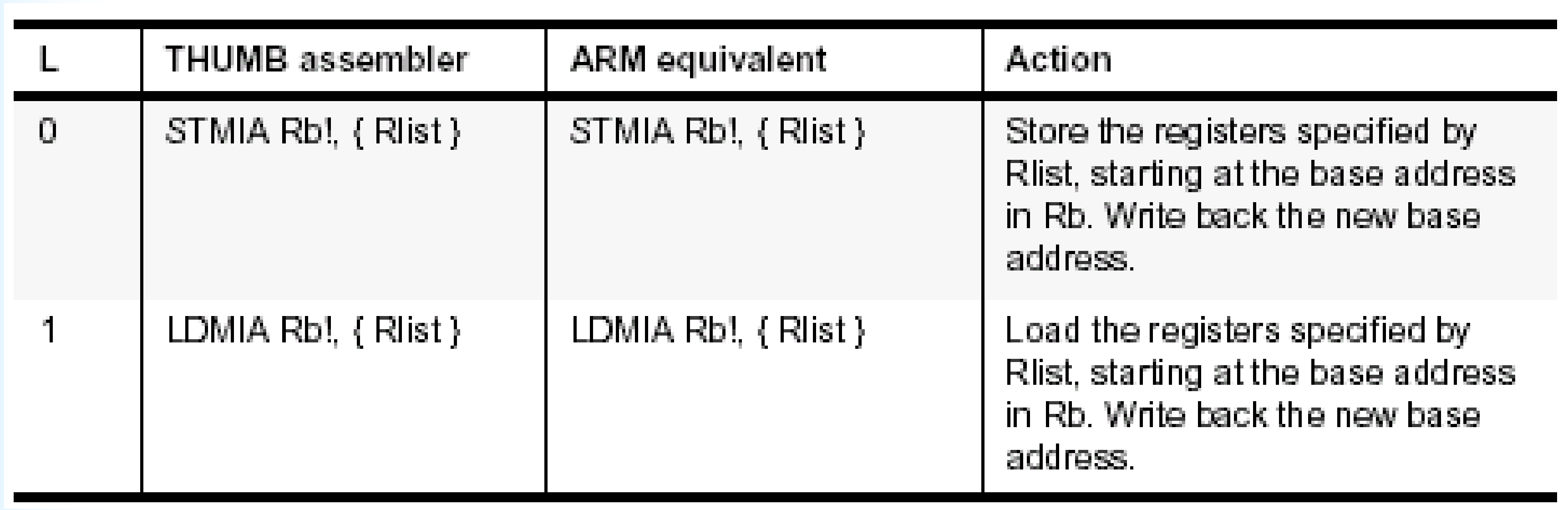
Multiple Load/Store

- `LDMIA     Rb!, {<reg list>}`
- `STMIA     Rb!, {<reg list>}`



Assembler Syntax

- `LDMIA      Rb!, {<reg list>}`
- `STMIA      Rb!, {<reg list>}`



L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

Example

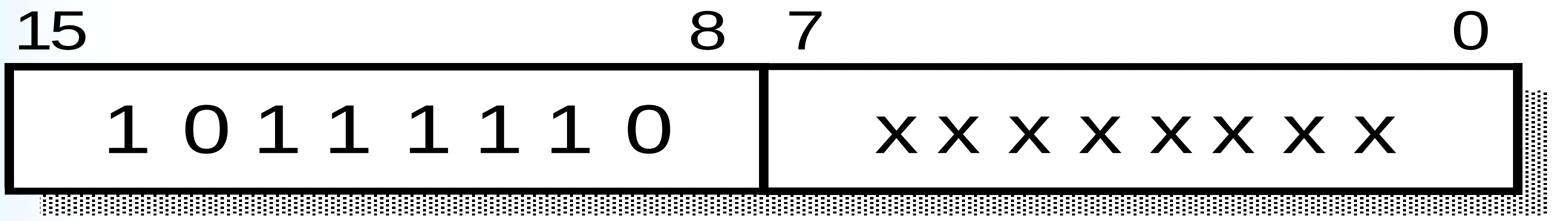
```
R1 = 0x00000001
R2 = 0x00000002
R3 = 0x00000003
R4 = 0x9000

STMIA    R4!, {R1, R2, R3}
```

```
mem32[0x9000] = 0x00000001
mem32[0x9004] = 0x00000002
mem32[0x9008] = 0x00000003
R4 = 0x900C
```

Thumb Breakpoint Binary Encoding

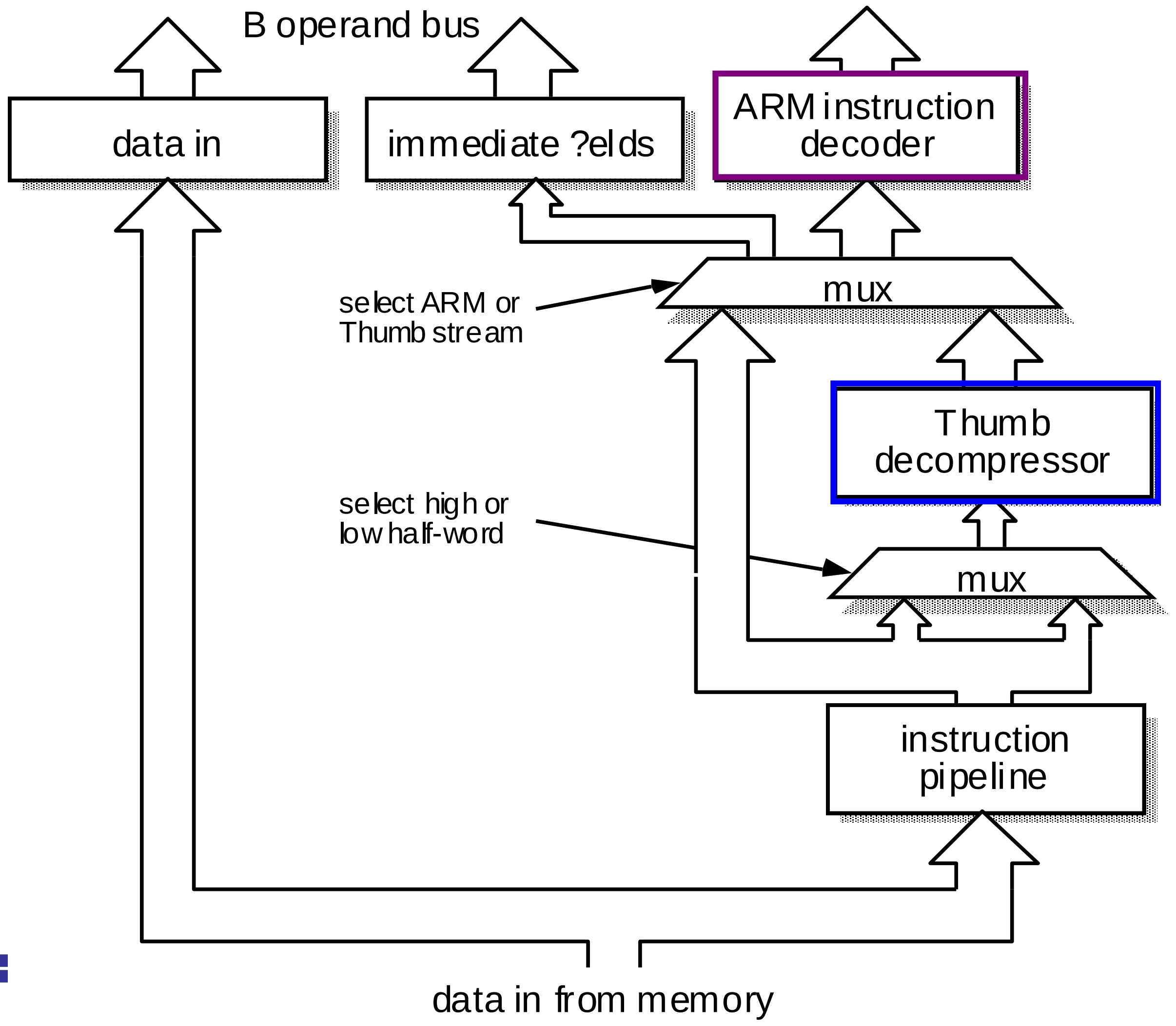
Assembler syntax: BKPT

ARM v5T

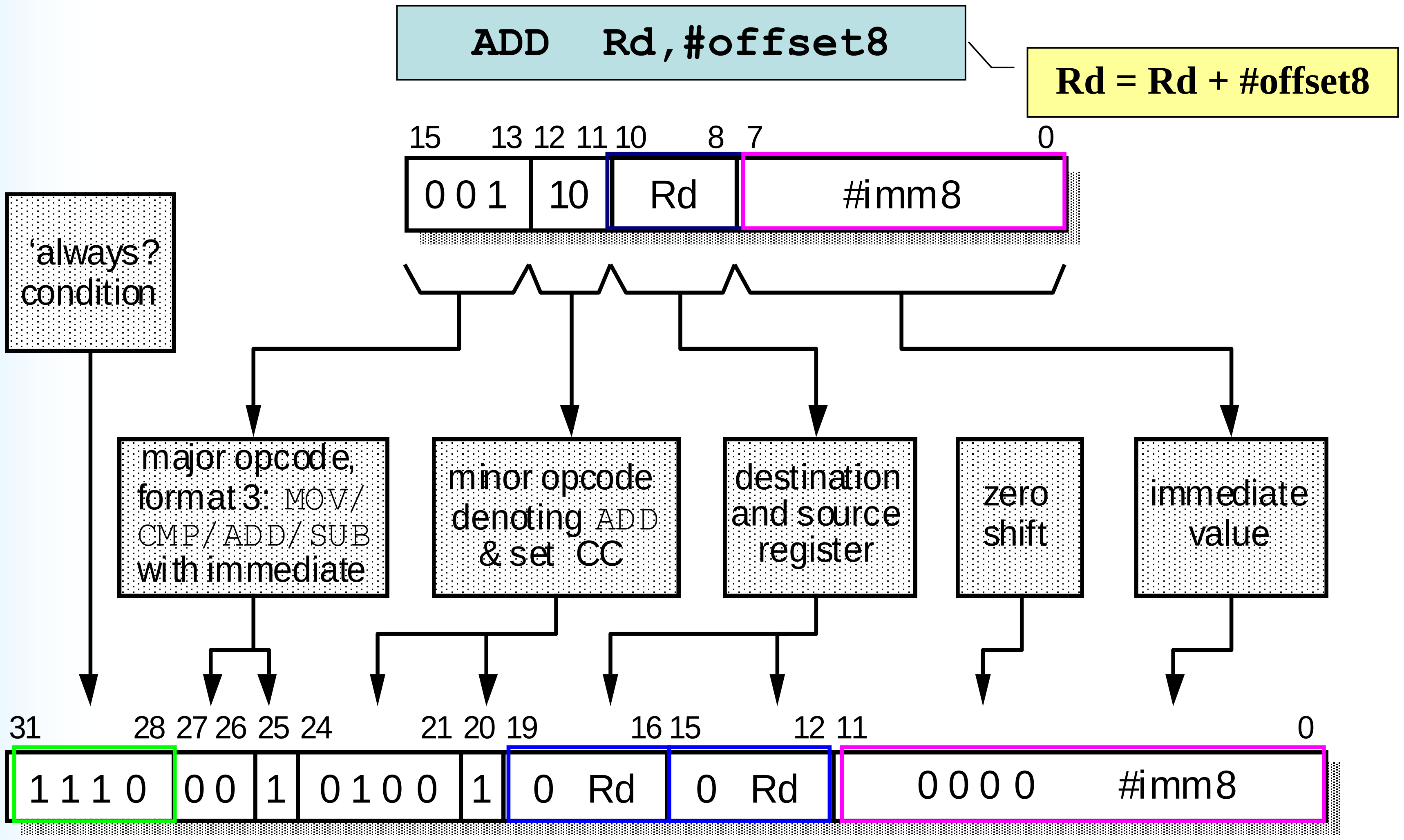


- The instruction causes the processor to take a **Prefetch Abort** exception
- A debug monitor program which has previously been installed on the **Prefetch Abort vector** can handle this exception

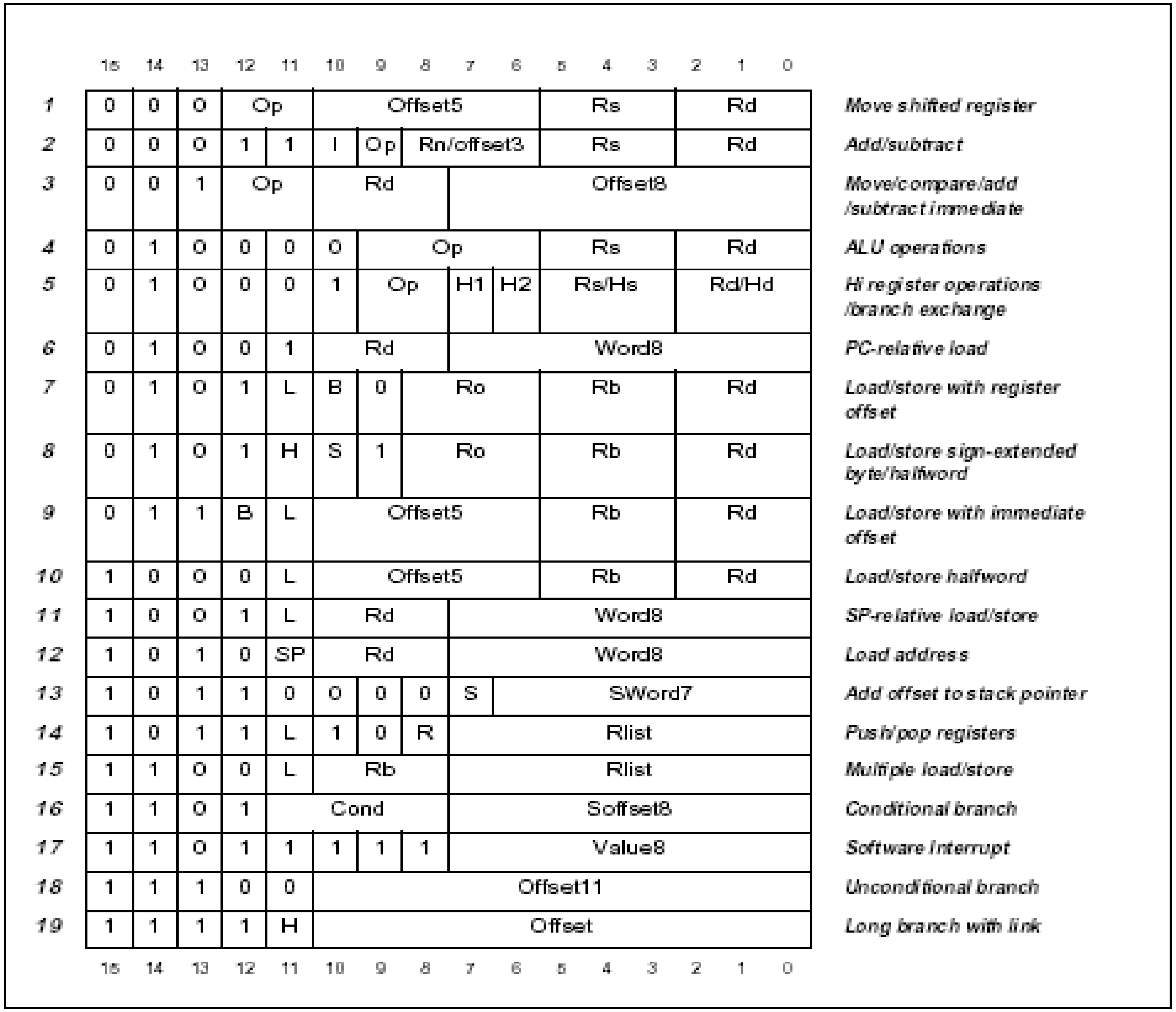
The Thumb Instruction Decompressor Organization



Thumb to ARM Instruction Mapping

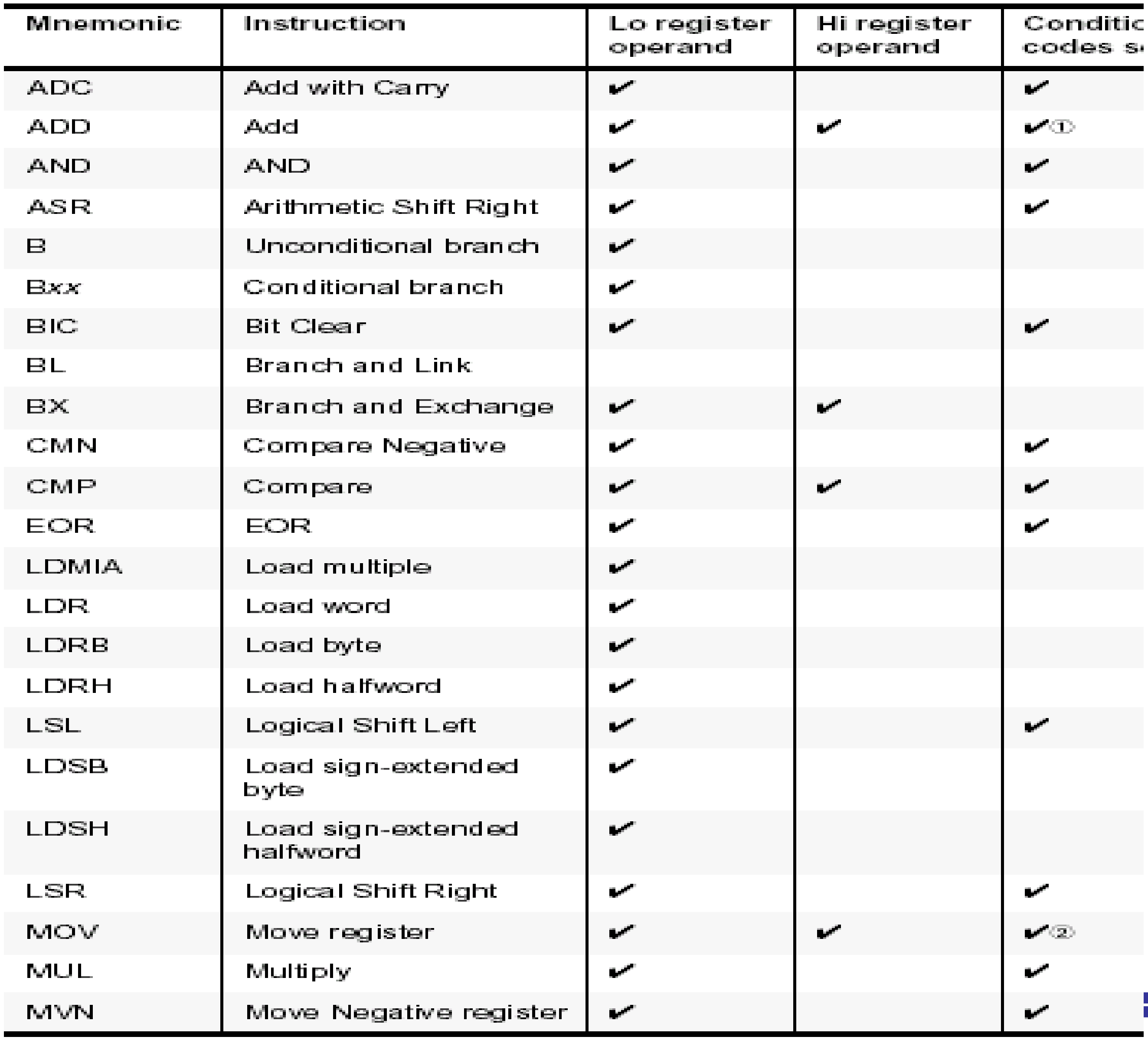


Format Summary



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset5					Rs			Rd			*Move shifted register*
2	0	0	0	1	1	I	Op	Rn/offset3			Rs			Rd			*Add/subtract*
3	0	0	1	Op		Rd			Offset8								*Move/compare/add /subtract immediate*
4	0	1	0	0	0	0	Op				Rs			Rd			*ALU operations*
5	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			Rd/Hd			*Hi register operations /branch exchange*
6	0	1	0	0	1	Rd			Word8								*PC-relative load*
7	0	1	0	1	L	B	0	Ro			Rb			Rd			*Load/store with register offset*
8	0	1	0	1	H	S	1	Ro			Rb			Rd			*Load/store sign-extended byte/halfword*
9	0	1	1	B	L	Offset5					Rb			Rd			*Load/store with immediate offset*
10	1	0	0	0	L	Offset5					Rb			Rd			*Load/store halfword*
11	1	0	0	1	L	Rd			Word8								*SP-relative load/store*
12	1	0	1	0	SP	Rd			Word8								*Load address*
13	1	0	1	1	0	0	0	0	S	SWord7							*Add offset to stack pointer*
14	1	0	1	1	L	1	0	R	Rlist								*Push/pop registers*
15	1	1	0	0	L	Rb			Rlist								*Multiple load/store*
16	1	1	0	1	Cond				Soffset8								*Conditional branch*
17	1	1	0	1	1	1	1	1	Value8								*Software Interrupt*
18	1	1	1	0	0	Offset11											*Unconditional branch*
19	1	1	1	1	H	Offset											*Long branch with link*
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Opcode Summary (1)



Mnemonic	Instruction	Lo register operand	Hi register operand	Conditic codes s
ADC	Add with Carry	✓		✓
ADD	Add	✓	✓	✓①
AND	AND	✓		✓
ASR	Arithmetic Shift Right	✓		✓
B	Unconditional branch	✓		
Bxx	Conditional branch	✓		
BIC	Bit Clear	✓		✓
BL	Branch and Link			
BX	Branch and Exchange	✓	✓	
CMN	Compare Negative	✓		✓
CMP	Compare	✓	✓	✓
EOR	EOR	✓		✓
LDMIA	Load multiple	✓		
LDR	Load word	✓		
LDRB	Load byte	✓		
LDRH	Load halfword	✓		
LSL	Logical Shift Left	✓		✓
LDSB	Load sign-extended byte	✓		
LDSH	Load sign-extended halfword	✓		
LSR	Logical Shift Right	✓		✓
MOV	Move register	✓	✓	✓②
MUL	Multiply	✓		✓
MVN	Move Negative register	✓		✓

Opcode Summary (2)

Mnemonic	Instruction	Lo register operand	Hi register operand	Condition codes set
NEG	Negate	✔		✔
ORR	OR	✔		✔
POP	Pop registers	✔		
PUSH	Push registers	✔		
ROR	Rotate Right	✔		✔
SBC	Subtract with Carry	✔		✔
STMIA	Store Multiple	✔		
STR	Store word	✔		
STRB	Store byte	✔		
STRH	Store halfword	✔		
SWI	Software Interrupt			
SUB	Subtract	✔		✔
TST	Test bits	✔		✔

① The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.

② The condition codes are unaffected by the format 5 version of this instruction.

- Backup

Thumb Applications

- **Thumb properties**
  - Thumb requires **70%** space of the ARM code
  - Thumb uses **40%** more instructions than the ARM code
  - With 32-bit memory, the ARM code is **40%** faster than the Thumb code
  - With 16-bit memory, the Thumb code is **45%** faster than the ARM code
  - Thumb uses **30%** less external memory power than ARM code

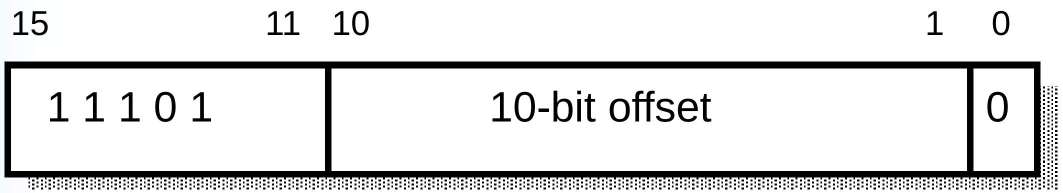
Thumb Systems

- **High-end 32-bit system**
  - Use Thumb code for non-critical routines
  - Save power or memory requirements
- **Low-end 16-bit system** (small on-chip 32-bit RAM)
  - Use ARM code for critical-routines
  - Use off-chip Thumb code for all non-critical routines

Thumb Branch Instructions (4)

- **BLX <label>   ;branch to Thumb target**
- **Supported in ARM v5T**
- **The operation of the instruction pair**
  - LR=PC + offset << 12   (BL, H=0)
  - PC=LR + (offset < 1) & 0xfffffffc   (H=1)
    - LR=next insn. address
    - The Thumb bit is cleared

1111111......11111100



`BLX <label>`