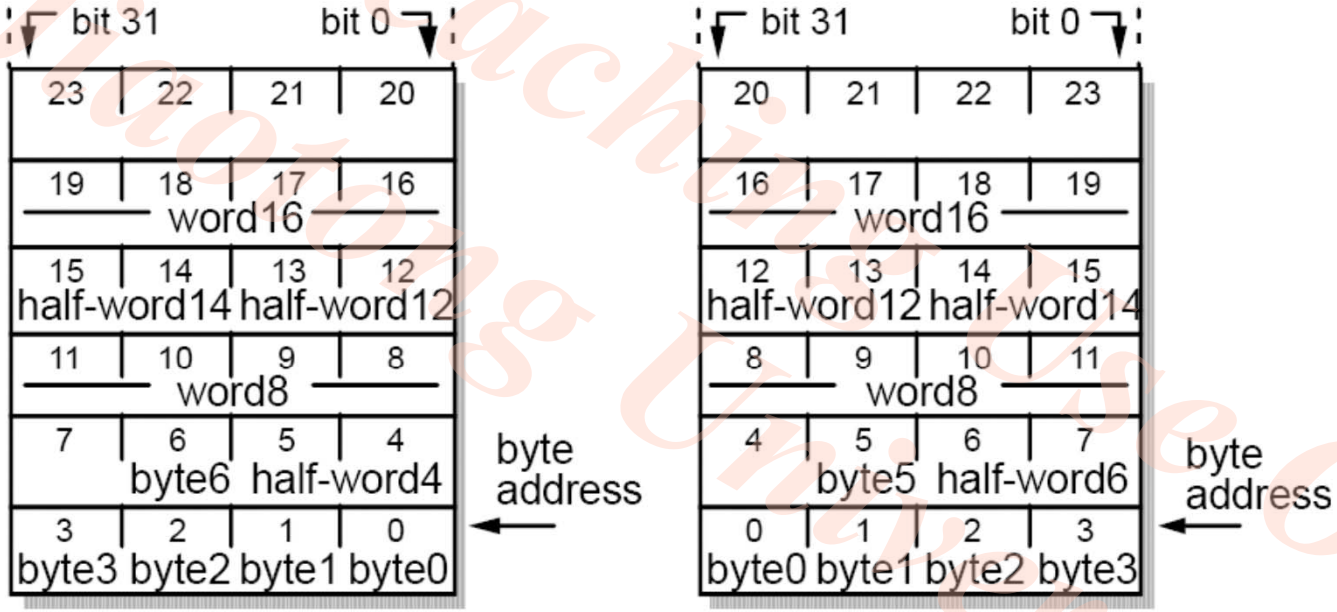
嵌入式系统设计与应用

第五章 ARM与Thumb指令集

西安交通大学电信学院
孙宏滨

1 深入ARM指令集

- 在以字节为单位寻址的存储器中，有两种方式来存储字，这根据最低有效字节与相邻较高有效字节相比是存在较低的还是较高的地址来划分。 **“On holy wars and a plea for peace”**



小端 *(a) Little-endian memory organization*

大端 *(b) Big-endian memory organization*

条件执行

- **ARM**指令集不同寻常的特征是每条指令都是条件执行。
- 条件域（**condition field**）占据**32**位指令域的高**4**位。

- 条件域共有**16**个值，每个值都根据**CPSR**中标志位**N**、**Z**、**C**和**V**的值来确定指令是执行还是跳过。

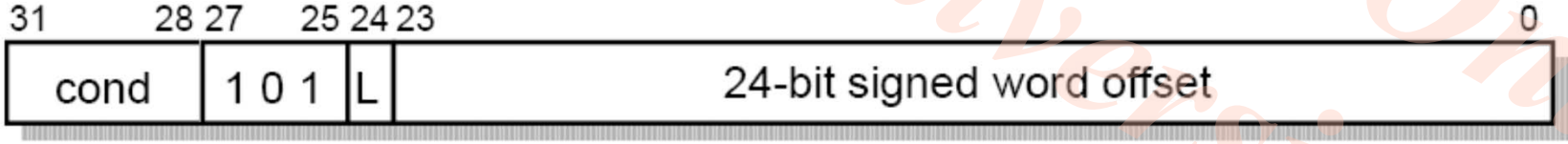
条件执行码

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
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

一般不应使用

转移及转移链接（B,BL）指令

- 汇编格式：　B{L}{<cond>}　　<target address>
- 转移和转移链接指令使处理器开始执行来自新地址的指令。地址计算方式：先对指令中定义的24位偏移量进行符号扩展，左移两位形成字的偏移，然后将它加到程序计数器。
- 转移指令的范围为+/-32MB
- 转移指令有位L（第24位）置位的链接形式，它将转移后下一条指令的地址传送到当前处理器模式下的链接寄存器（r14）。这一般用于实现子程序的调用，返回时将链接寄存器的内容拷贝回PC。



B&BL指令示例（1）

1. 无条件跳转：

```
        B       Label
        ...
Label   ...
```

2. 执行循环10次：

```
        MOV     r0, #10
        ...
LOOP    ...
        SUBS    r0, #1
        BNE     LOOP
```

B&BL指令示例（2）

3. 调用子程序：

```
        ...
        BL      SUB
        ...
SUB     ...
        MOV   pc,    r14
```

4. 条件子程序调用：

```
        ...
        CMP    r0,  #5
        BLLT   SUB1
        BLGE   SUB2
```

转移交换指令

- 汇编格式：　　**BX{<cond>}　　Rm**

- 转移交换指令用于支持**Thumb**（**16**位）指令集的**ARM**芯片，是一种将处理器切换到执行**Thumb**指令或返回到**ARM**的机制。一条类似的**Thumb**指令可以使处理器切换回**32**位**ARM**指令。

31 – 28	27 – 6	5	4	3 – 0
cond	0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0	L	1	Rm

- If Rm[0] is 1, the processor switches to execute Thumb instruction.
- If Rm[0] is 0, the processor continues executing ARM instruction.

示例（1）

(1) An unconditional jump

```
        BX      r0      ; branch to address in r0
                        ; enter Thumb state if r0[0]=1
```

(2) A call to a Thumb subroutine

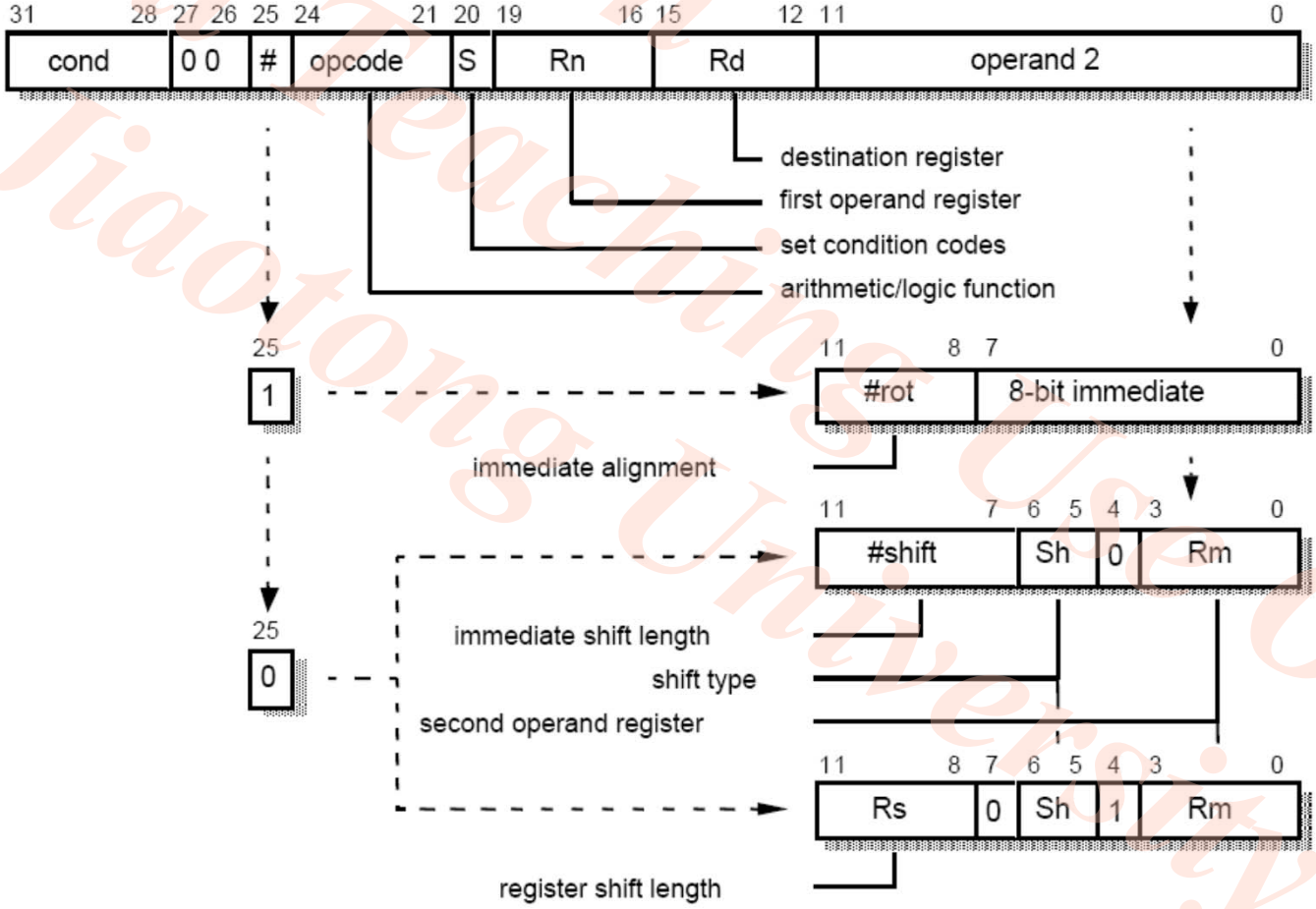
```
        CODE32          ; ARM code follows
        ...
        BLX     TSUB    ; call Thumb subroutine
        ...
        CODE16          ; start of Thumb code
TSUB    ...             ; Thumb subroutine
        BX      r14     ; return to ARM code
```

示例（2）

```
        AREA ThumbSub, CODE, READONLY   ; Name this block of code
        ENTRY                           ; Mark first instruction to execute
        CODE32                          ; Subsequent instructions are ARM
header  ADR     r0, start + 1           ; Processor starts in ARM state,
        BX      r0                      ; so small ARM code header used
                                        ; to call Thumb main program
        CODE16                          ; Subsequent instructions are Thumb
start
        MOV     r0, #10                 ; Set up parameters
        MOV     r1, #3
        BL      doadd                   ; Call subroutine
stop
        MOV     r0, #0x18               ; angel_SWIreason_ReportException
        LDR     r1, =0x20026            ; ADP_Stopped_ApplicationExit
        SWI     0xAB                    ; Thumb semihosting SWI
doadd
        ADD     r0, r0, r1              ; Subroutine code
        MOV     pc, lr                  ; Return from subroutine
        END                             ; Mark end of file
```

数据处理指令

- 汇编格式：**<op>{<cond>}{S} Rd, Rn, #<32bit immediate>**
  <op>{<cond>}{S} Rd, Rn, Rm, {<shift>}



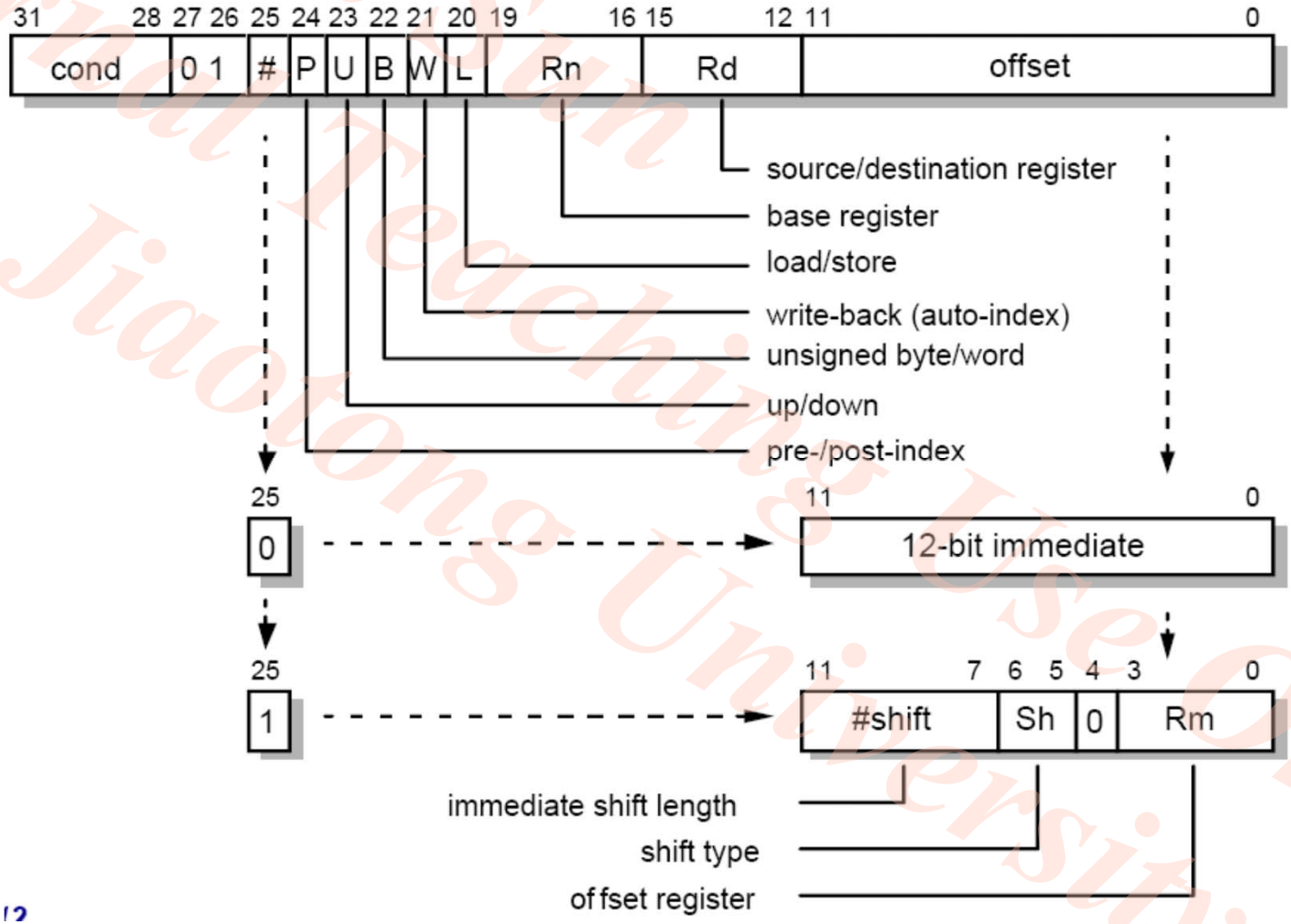
ARM数据处理指令

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

单字与无符号字节传送指令（1）

- 前变址指令格式如下：
  - **LDR|STR {<cond>}{B}　　Rd, [Rn, <offset>]{!}**
- 后变址指令格式如下：
  - **LDR|STR {<cond>}{B}{T}　　Rd, [Rn], <offset>**
- 一种有用的相对**PC**的形式（由汇编器计算所需立即数）：
  - **LDR|STR {<cond>}{B}　　Rd, LABEL**

单字与无符号字节传送指令（2）



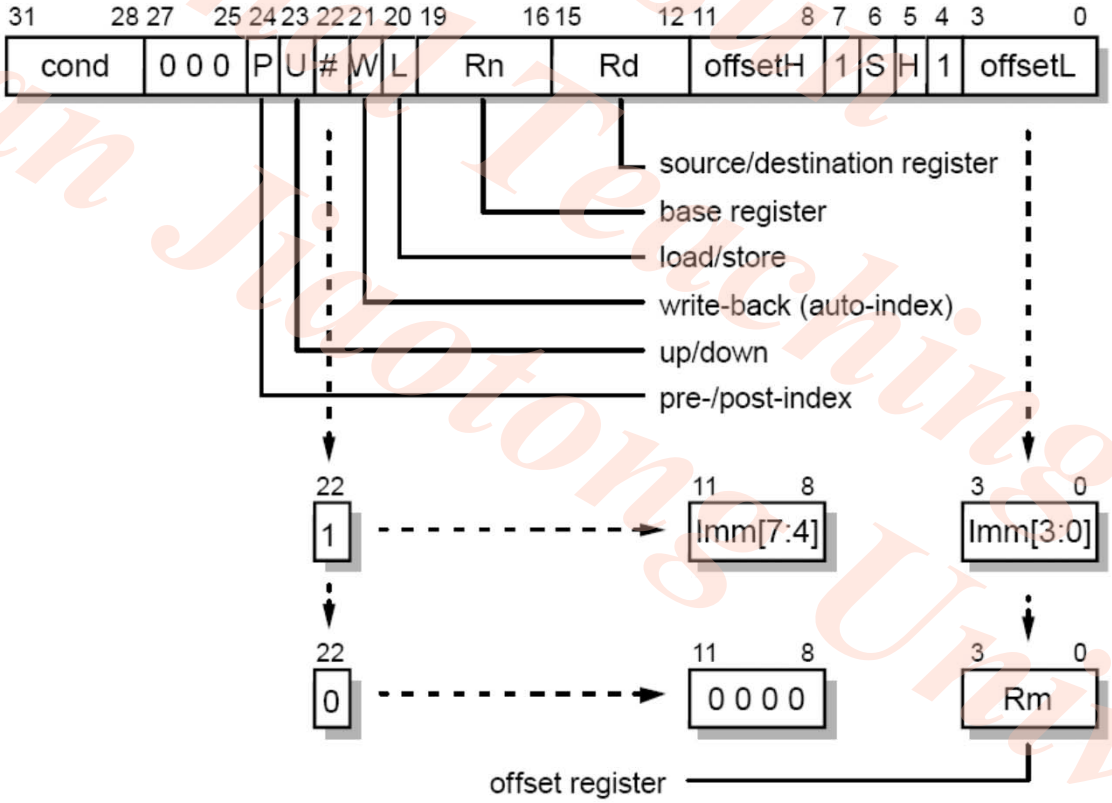
半字和有符号字节传送指令（1）

- 前变址格式：
  - **LDR|STR {<cond>}H|SH|SB Rd, [Rn, <offset>]{!}**

- 后变址格式：
  - **LDR|STR {<cond>}H|SH|SB Rd, [Rn], <offset>**

  - **<offset>是<+/-8位立即数>或#+/-Rm**

半字和有符号字节传送指令（2）

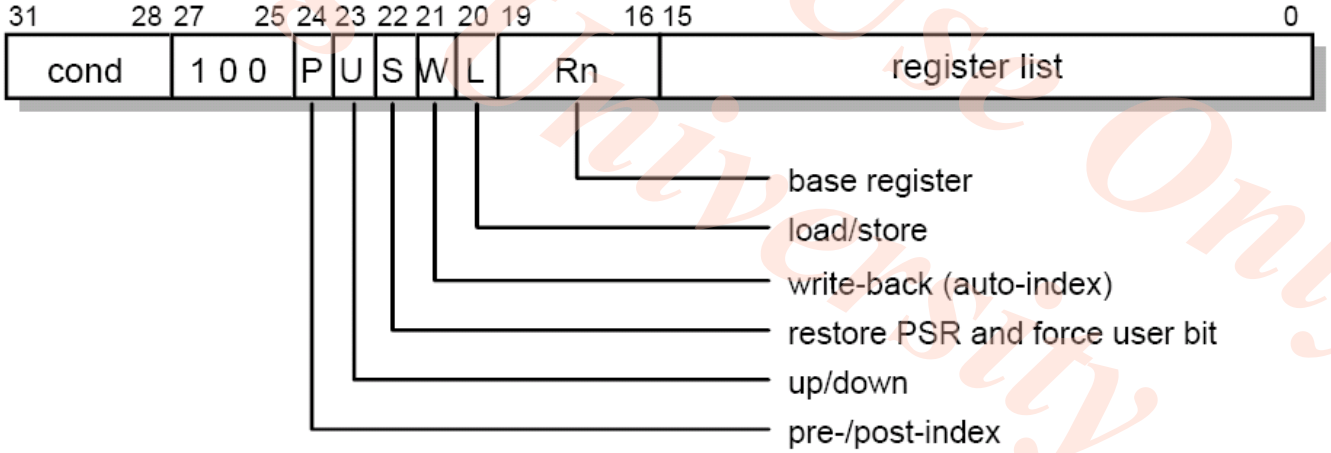


S	H	Data type
1	0	Signed byte
0	1	Unsigned half-word
1	1	Signed half-word

多寄存器传送指令

- 指令格式：
  - **LDM|STM {<cond>}<add mode> Rn{!}, <register>**

		Ascending Full	Ascending Empty	Descending Full	Descending Empty
Increment	Before	STMIB STMFA			LDMIB LDMED
	After		STMIA STMEA	LDMIA LDMFD	
Decrement	Before		LDMDB LDMEA	STMDB STMFD	
	After	LDMDA LDMFA			STMDA STMED



2 Thumb指令集

- Thumb指令集是针对代码密度的问题而提出的。它可以看作是ARM指令压缩形式的子集。
- 所有的Thumb指令都有相对应的ARM指令，而Thumb的编程模型也对应于ARM的编程模型。
- 在ARM指令流水线中实现Thumb指令须先进行动态解压缩，然后再把它作为标准ARM指令来执行。
- Thumb是一个不完整的体系结构。它只支持通用功能，在必要时需要借助于完整的ARM指令集。
- 支持Thumb指令集的ARM处理器也可以执行标准的32位ARM指令集。不是所有的ARM处理器都可以执行Thumb指令。

CPSR中的Thumb指示位

- 指令流的解释取决于CPSR寄存器的第5位，即位T。若T置位，则认为指令流为16位的Thumb指令；否则为标准的ARM指令。

- 并不是所有的ARM处理器都支持Thumb指令。只有在命名中有字母T的才支持。例如ARM7TDMI。
  - Thumb：16位压缩指令
  - On-chip Debug support：处理器可以暂停相应调试请求
  - An enhance Multiplier：高性能，64位结果输出
  - Embedded ICE hardware：支持片上断点和观察点设置

进入和退出Thumb模式

- 进入Thumb模式：转向执行Thumb指令的通常方法是执行一条交换转移指令BX(Branch and Exchange)

```
Example_1:
            MOV     R0, #0x40
            BX      R0
Example_2:
            MOV     R0, #0x43
            BX      R0
```

若BX指令指定的寄存器的最低位为1，则将T置位，并将PC切换为寄存器其余数据位给出的地址。

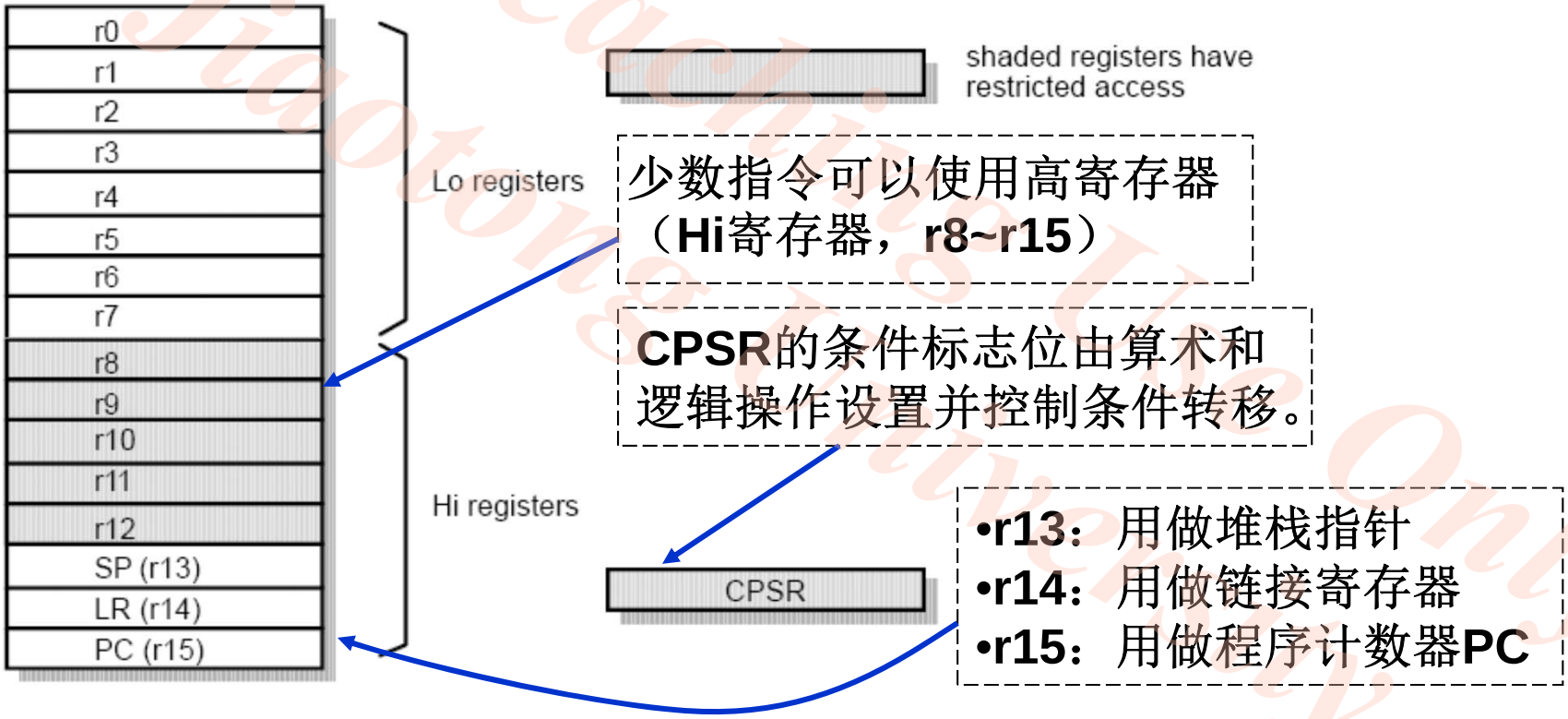
- 退出Thumb模式：
  - 执行Thumb BX指令可以显式地返回到ARM指令集
  - 由于进入异常总是在ARM模式进行，因此，任何时候发生异常都能隐含地返回到ARM指令集

Thumb系统

- 一个典型的嵌入式系统会在ARM核所在的芯片上集成一个小的、高速的32位存储器，把有速度要求的关键子程序（如数字信号处理算法）以ARM代码形式保存在这个存储器中。大多数对速度没有要求的程序存储在片外16位ROM中。

Thumb编程模型

- **Thumb**指令集对低（**Lo**）**8**个通用寄存器**r0~r7**具有全部访问权限，并扩展寄存器**r13~r15**的使用以作特殊用途。
- 其他寄存器（**r8~r12**及**CPSR**）只能作有限访问。

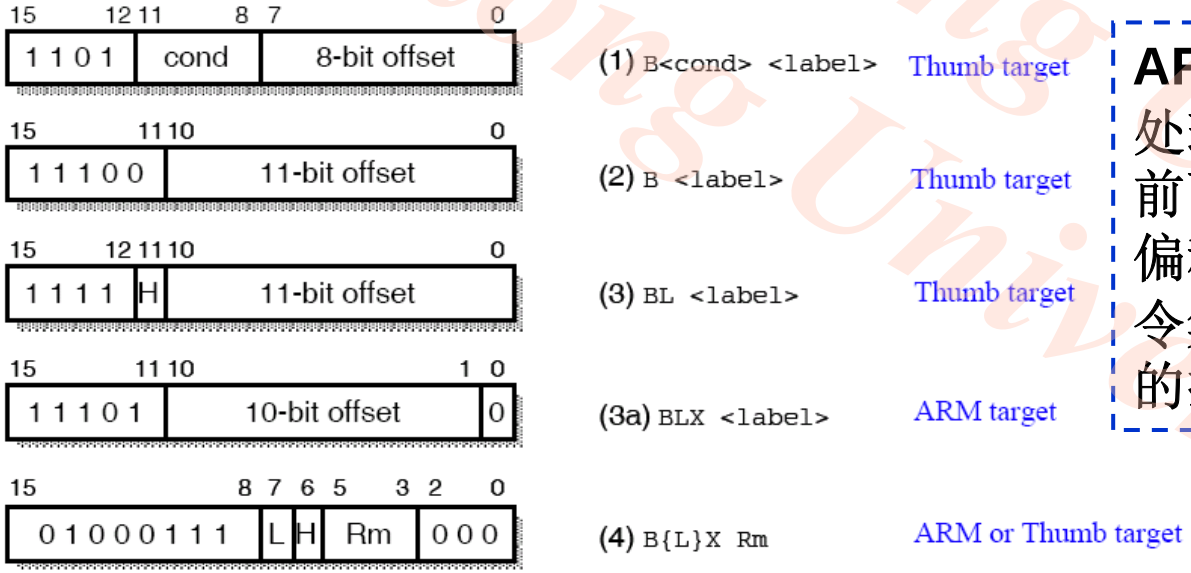


Thumb与ARM的异同

- Thumb-ARM相似处：Thumb指令都是16位，都有相对应的ARM指令，因此继承了ARM指令集的许多特点。
  - Load-Store结构，有数据处理、数据传送及流控制指令
  - 支持8位字节、16位半字和32位字数据类型
  - 32位无分段存储器
- Thumb-ARM差异处：为实现16位指令长度，丢弃了ARM指令集一些特性。
  - 大多数Thumb指令无条件执行(所有ARM指令都是条件执行)
  - 大多数Thumb数据处理指令采用2地址格式（目的寄存器与一个源寄存器相同）（除64位乘法指令外，ARM数据处理指令采用3地址格式。）
  - 由于采用高密度编码，Thumb指令没有ARM指令格式规则

Thumb转移指令

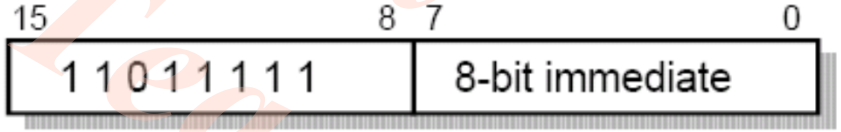
- **ARM**指令有一个大的（**24**位）偏移域，不可能在**16**位**Thumb**指令格式中表示。因此，**Thumb**指令集通过多种方法实现其子功能。
- 转移指令的典型用法包括：
  - 短距离条件转移指令可用于控制循环的退出
  - 中等距离的无条件转移指令用于实现**goto**功能



ARM指令集用同一条指令处理所有这些情况，但在前两种情况下浪费了**24**位偏移的很多位。**Thumb**指令集对每种情况采用不同的指令模式，因而更高效。

Thumb软中断指令

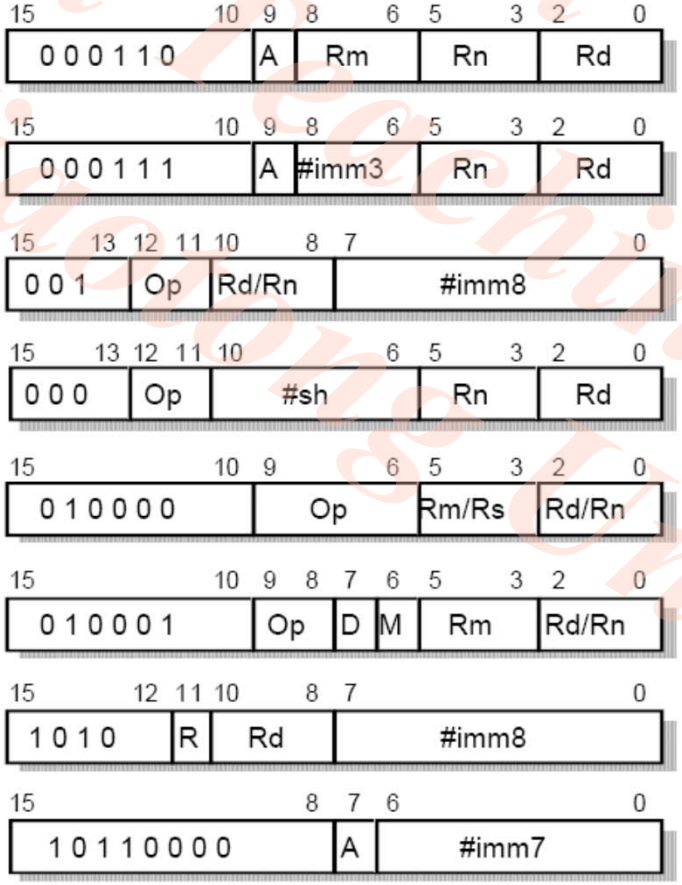
- Thumb软中断指令的行为与ARM等价指令完全相同。进入异常的指令使微处理器进入ARM执行状态。



- Thumb软中断指令将引起下列动作：
  - 将下一条Thumb指令的地址保存到r14_svc；
  - 将CPSR寄存器保存到SPSR_svc；
  - 微处理器关闭IRQ，将Thumb位清0，并通过修改CPSR的相关位进入监控模式；
  - 强制将PC值置为地址0x08。
- 然后进入ARM指令SWI的处理程序。正常的返回指令将恢复Thumb的执行状态。

Thumb数据处理指令（1）

- **Thumb**数据处理指令包括一组高度优化且相当复杂的指令，范围涵盖编译器通常需要的大多数操作。



(1) ADD|SUB Rd,Rn,Rm

(2) ADD|SUB Rd,Rn,#imm3

(3) <Op> R d/Rn ,#imm8

(4) LSL|LSR|ASR Rd,Rn,#shift

(5) <Op> Rd/Rn,Rm/Rs

(6) ADD|CMP|MOV Rd/Rn,Rm

(7) ADD Rd,SP|PC,#imm8

(8) ADD|SUB SP,SP,#imm7

Thumb数据处理指令（2）

- 使用低8个通用寄存器（r0~r7）的指令：

ARM instruction		Thumb instruction	
MOVS	Rd, #<#imm8>	;MOV	Rd, #<#imm8>
MVNS	Rd, Rm	;MVN	Rd, Rm
CMP	Rn, #<#imm8>	;CMP	Rn, #<#imm8>
CMP	Rn, Rm	;CMP	Rn, Rm
CMN	Rn, Rm	;CMN	Rn, Rm
TST	Rn, Rm	;TST	Rn, Rm
ADDS	Rd, Rn, #<#imm3>	;ADD	Rd, Rn, #<#imm3>
ADDS	Rd, Rd, #<#imm8>	;ADD	Rd, Rd, #<#imm8>
ADDS	Rd, Rn, Rm	;ADD	Rd, Rn, Rm
ADCS	Rd, Rn, Rm	;ADC	Rd, Rm
SUBS	Rd, Rn, #<#imm3>	;SUB	Rd, Rn, #<#imm3>
SUBS	Rd, Rd, #<#imm8>	;SUB	Rd, Rd, #<#imm8>
SUBS	Rd, Rn, Rm	;SUB	Rd, Rn, Rm
SBCS	Rd, Rn, Rm	;SBC	Rd, Rm
RSBS	Rd, Rn, #0	;NEG	Rd, Rn

Thumb数据处理指令（3）

- 使用低8个通用寄存器（r0~r7）的指令：

ARM instruction		Thumb instruction	
MOVS	Rd, Rm, LSL #<#sh>	;LSL	Rd, Rm, #<#sh>
MOVS	Rd, Rd, LSL Rs	;LSL	Rd, Rs
MOVS	Rd, Rm, LSR #<#sh>	;LSR	Rd, Rm, #<#sh>
MOVS	Rd, Rd, LSR Rs	;LSR	Rd, Rs
MOVS	Rd, Rm, ASR #<#sh>	;ASR	Rd, Rm, #<#sh>
MOVS	Rd, Rd, ASR Rs	;ASR	Rd, Rs
MOVS	Rd, Rd, ROR Rs	;ROR	Rd, Rs
ANDS	Rd, Rd, Rm	;AND	Rd, Rm
EORS	Rd, Rd, Rm	;EOR	Rd, Rm
ORRS	Rd, Rd, Rm	;ORR	Rd, Rm
BICS	Rd, Rd, Rm	;BIC	Rd, Rm
MULS	Rd, Rd, Rm	;MUL	Rd, Rm

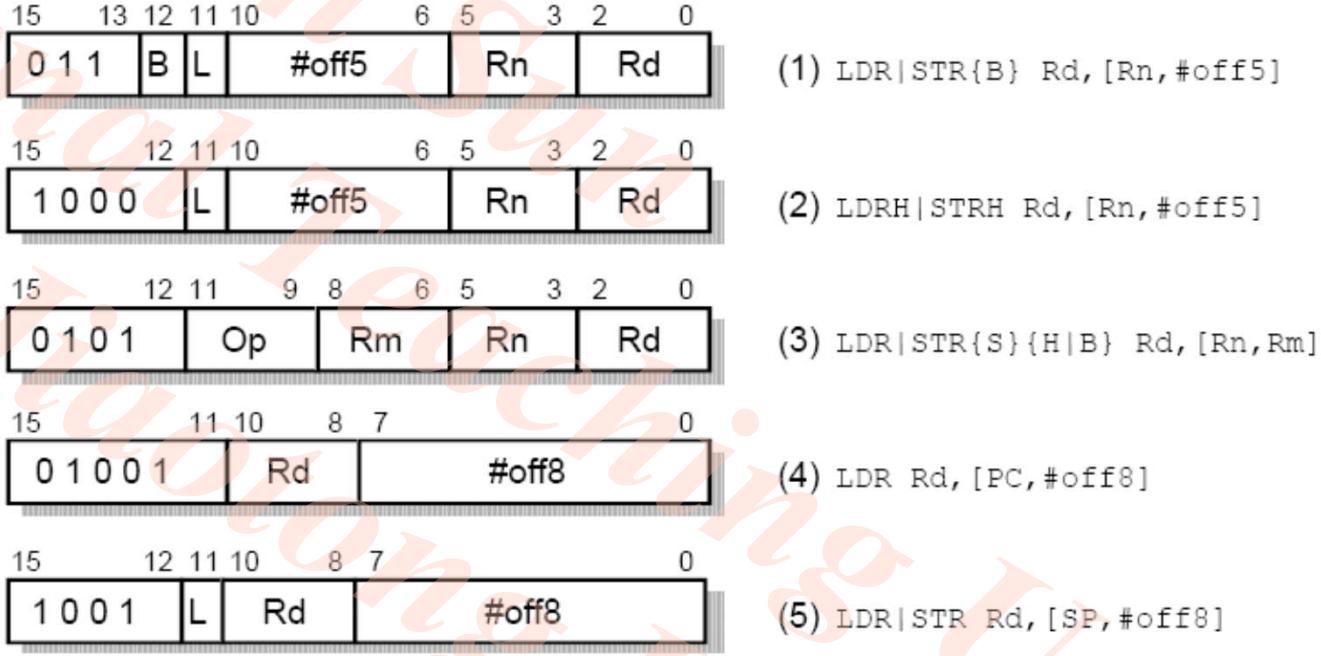
Thumb数据处理指令（4）

- 使用高8个寄存器（r8~r15）的指令。在有些情况下结合低8个寄存器使用。

ARM instruction		Thumb instruction	
ADD	Rd, Rd, Rm	;ADD	Rd, Rm (1/2 Hi regs)
CMP	Rn, Rm	;CMP	Rn, Rm (1/2 Hi regs)
MOV	Rd, Rm	;MOV	Rd, Rm (1/2 Hi regs)
ADD	Rd, PC, #<#imm8>	;ADD	Rd, PC, #<#imm8>
ADD	Rd, SP, #<#imm8>	;ADD	Rd, SP, #<#imm8>
ADD	SP, SP, #<#imm7>	;ADD	SP, SP, #<#imm7>
SUB	SP, SP, #<#imm7>	;SUB	SP, SP, #<#imm7>

- 所有对低8个寄存器操作的数据处理指令都更新条件码位（等价的ARM指令位S置位）。
- 对高8个寄存器操作的指令不改变条件码位。CMP指令除外，它只改变条件码。
- 指令中“1/2 Hi regs”表示至少有1个操作数是高8个寄存器。
- #imm{3,7,8}分别表示3位、7位和8位立即数域。#sh表示5位移位数域

Thumb单寄存器数据传送指令（1）



- 这些指令是从**ARM**单寄存器传送指令中精心导出的子集，并且与等价的**ARM**指令有严格相同的语义。
- 在所有的指令中，对偏移量需要根据数据类型按比例调整。例如，**5**位偏移量的范围在字节**Load**和**Store**指令中是**32**字节，在半字指令中**64**字节，在字指令中是**128**字节。

Thumb单寄存器数据传送指令（2）

Examples

```
LDR     r3,[r5,#0]
STRB    r0,[r3,#31]
STRH    r7,[r3,#16]
LDRB    r2,[r4,#label-{PC}]
```

Incorrect examples

```
LDR     r13,[r5,#40]     ; high registers not allowed

STRB    r0,[r3,#32]      ; 32 is out of range for byte transfers

STRH    r7,[r3,#15]      ; offsets for halfword transfers must be even

LDRH    r6,[r0,#-6]      ; negative offsets not supported
```

Thumb单寄存器数据传送指令（3）

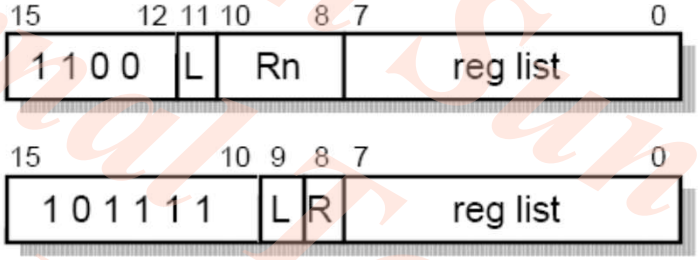
Examples

```
LDR     r2,[pc,#1016]
LDR     r5,localdata
LDR     r0,[sp,#920]
STR     r1,[sp,#20]
```

Incorrect examples

```
LDR     r13,[pc,#8]    ; Rd must be in range r0-r7

STR     r7,[pc,#64]    ; there is no pc-relative STR instruction

STRH    r0,[sp,#16]    ; there are no pc- or sp-relative
                       ; halfword or byte transfers

LDR     r2,[pc,#81]    ; immediate must be a multiple of four

LDR     r1,[pc,#-24]   ; immediate must not be negative

STR     r1,[sp,#1024]  ; maximum immediate value is 1020
```

Thumb多寄存器数据传送指令（1）



(1) LDMIA|STMIA Rn!, {<reg list>}

(2) POP|PUSH {<reg list>{,R}}

- 指令的块拷贝形式使用**LDMIA**和**STMIA**寻址模式。**Lo**寄存器中的任何一个可以作为基址寄存器。寄存器列表可以是这些寄存器的任意子集，但不应包括基址寄存器。
- 堆栈使用**SP(r13)**作为基址寄存器。堆栈的模式也固定为满栈递减。寄存器列表除了可以是**8**个**Lo**寄存器外，链接寄存器**LR(r14)**可出现在**PUSH**指令中，**PC(r15)**可出现在**POP**指令中。

Examples

```
LDMIA   r3!, {r0,r4}
LDMIA   r5!, {r0-r7}
STMIA   r0!, {r6,r7}
STMIA   r3!, {r3,r5,r7}
```

Incorrect examples

```
LDMIA   r3!,{r0,r9} ; high registers not allowed

STMIA   r5!, {}     ; must be at least one register
                    ; in list

STMIA   r5!,{r1-r6} ; value stored from r5 is unpredictable
```

Thumb多寄存器数据传送指令（2）

Examples

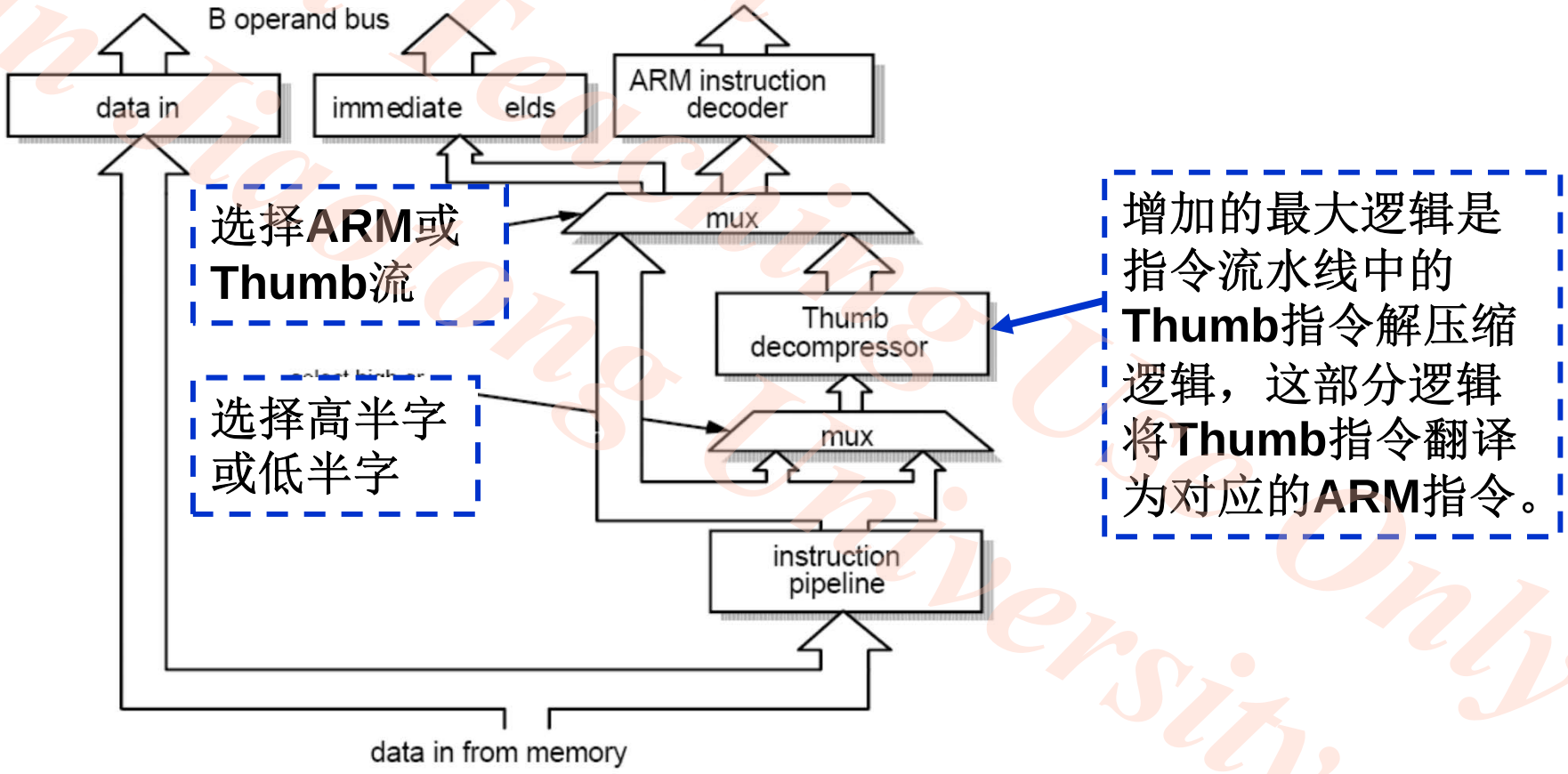
```
PUSH    {r0,r3,r5}
PUSH    {r1,r4-r7}  ; pushes r1, r4, r5, r6, and r7
PUSH    {r0,LR}
POP     {r2,r5}
POP     {r0-r7,pc}  ; pop and return from subroutine
```

Incorrect examples

```
PUSH    {r3,r5-r8}  ; high registers not allowed
PUSH    {}          ; must be at least one register in list
PUSH    {r1-r4,pc}  ; cannot push the pc
POP     {r1-r4,LR}  ; cannot pop the LR
```

Thumb的实现

- 对3级流水线ARM处理器的大部分逻辑作相对小的改动就可以实现Thumb指令集（5级流水线的实现要复杂些）。



Thumb指令映射（1）

- Thumb解压缩器逻辑将16位Thumb指令静态地转换为等价的32位ARM指令。这包括：
  - 主操作码和次操作码的查表转换
  - 3位寄存器指示符（specifier）零扩展成4位寄存器指示符
  - 所需的其他域的映射
- 例子：

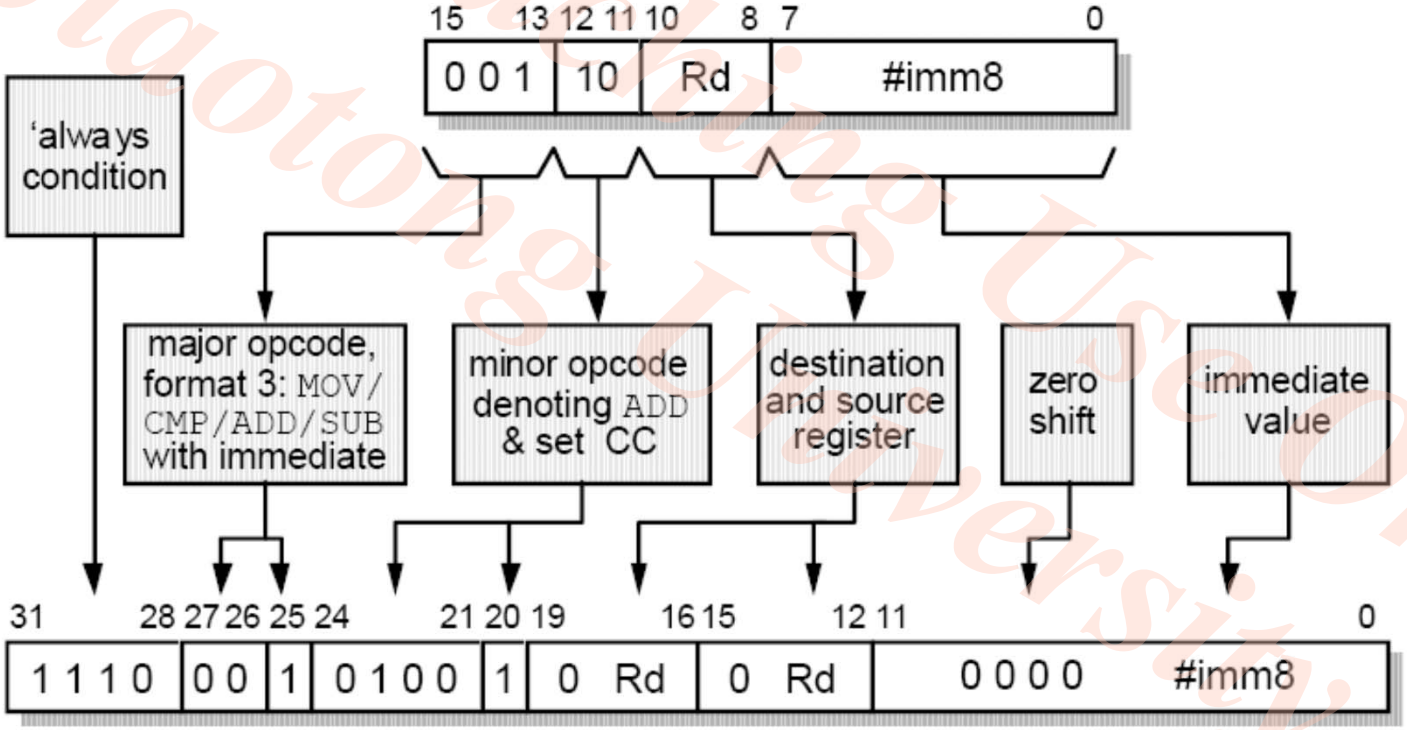
  Thumb：ADD　Rd, #imm8

  ARM:　ADDS Rd, Rd, #imm8

- 指令解压缩逻辑的简单性对Thumb指令集的效率是非常重要的。如果Thumb解压缩逻辑复杂、速度低并且功耗大，那么Thumb就没有什么价值了。

Thumb指令映射（2）

- **Thumb**：ADD　Rd, #imm8;　　**ARM**：ADDS Rd, Rd, #imm8
- 转移指令是唯一条件执行T指令，其他T指令在转换时使用条件“always”
- 是否修改CPSR的条件码在T操作码中隐含指定，但A指令中要明确指定
- 可通过重复寄存器指示符将T的2地址指令格式转换为A的3地址指令格式



Thumb的应用

- Thumb的特点
  - Thumb代码所需空间为ARM代码的70%
  - Thumb代码使用的指令数比ARM代码多40%
  - 用32位存储器，ARM代码比Thumb代码快40%
  - 用16位存储器，Thumb代码比ARM代码快45%
  - 使用Thumb代码，外部存储器功耗比ARM代码少30%

若性能最重要，则系统使用32位存储器和运行ARM代码；若成本及功耗更重要，则最好选择16位存储器系统及Thumb代码。若两者结合使用，会在两方面取得最好的效果。

- Thumb系统
  - 高端的32位ARM系统可以用Thumb代码实现特定的非关键程序，以节省功耗或降低对存储器的需求。
  - 低端的16位系统可以有小规模的32位片上RAM供运行ARM代码的关键程序使用，所有非关键程序使用片外Thumb代码。

ARM体系结构的发展历程（3）

v4	v5	v6	v7
Halfword and signed halfword / byte support System mode Thumb instruction set (v4T)	Improved interworking CLZ Saturated arithmetic DSP MAC instructions Extensions: Jazelle (5TEJ)	SIMD Instructions Multi-processing v6 Memory architecture Unaligned data support Extensions: Thumb-2 (6T2) TrustZone® (6Z) Multicore (6K) Thumb only (6-M)	Thumb-2 Architecture Profiles 7-A - Applications 7-R - Real-time 7-M - Microcontroller

- **Note that implementations of the same architecture can be different**
  - Cortex-A8 - architecture v7-A, with a 13-stage pipeline
  - Cortex-A9 - architecture v7-A, with an 8-stage pipeline

3 应用处理器 Application Processor

- 华邦 Winbond → 芯唐 Nuvoton
- W90P710 → NUC710