Microprocessors

Tuba Ayhan

MEF University

ARM Processor – Instructions

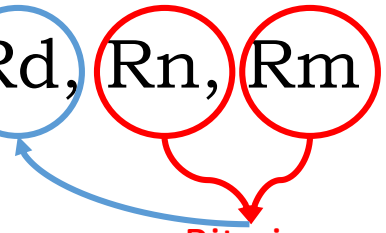
Logic and Test Instructions

Computer Organization and Embedded Systems, Hamacher et. al

Logic Instructions

- Logic operations AND, OR, XOR, and Bit-Clear
- Instructions AND, ORR, EOR, and BIC

```
AND Rd, Rn, Rm
```



Bitwise and two operands

Write here

- BIC complements each bit in operand Rm before ANDing them with the bits in register Rn.
- Example: R0 = 02FA62CA R1 = 0000FFFF

```
AND R0, R0, R1      R0 : 000062CA
BIC R0, R0, R1      R0 : 02FA0000
```

Example - *packed-BCD*

- Task:
  - Two decimal digits represented in ASCII code are located in the memory at byte locations LOC and LOC + 1
  - Represent each of these digits in the 4-bit BCD code and store both of them in a single byte location PACKED
- Solution:
  - Extract the low-order four bits in LOC and LOC + 1 and concatenate them into the single byte at PACKED.

Table 1.1 The 7-bit ASCII code.

Bit positions	Bit positions 654				
3210	000	001	010	011	100
0000	NUL	DLE	SPACE	0	@
0001	SOH	DC1	!	1	A
0010	STX	DC2	”	2	B
0011	ETX	DC3	#	3	C
0100	EOT	DC4	$	4	D
0101	ENQ	NAK	%	5	E
0110	ACK	SYN	&	6	F
0111	BEL	ETB	’	7	G
1000	BS	CAN	(	8	H
1001	HT	EM	)	9	I
1010	LF	SUB	*	:	J
1011	VT	ESC	+	;	K
1100	FF	FS	,	<	L
1101	CR	GS	-	=	M
1110	SO	RS	.	>	N
1111	SI	US	/	?	O

Digit-Packing Program

```
LDR     R0, =LOC            Load address LOC into R0.
LDRB    R1, [R0]            Load ASCII characters
LDRB    R2, [R0, #1]           into R1 and R2.
AND     R2, R2, #&F         Clear high-order 28 bits of R2.
ORR     R2, R2, R1, LSL #4  Shift contents of R1 left,
                               perform logical OR with
                               contents of R2, and place
                               result into R2.
STRB    R2, PACKED          Store packed BCD digits
                               into PACKED.
```

ASCII-Byte1	ASCII-Byte2	ASCII-Byte3	ASCII-Byte4
Loaded to R1	Loaded to R2		

- Load the address LOC into register R0.
- Load two ASCII bytes to R1 and R2.
- Clear the high-order 28-bits.
  - '&' character : hexadecimal immediate value.
- Shift the first BCD digit in R1 to the left four positions, places it to the left of the second BCD digit.

Digit-Packing Program

```
LDR     R0, =LOC            Load address LOC into R0.
LDRB    R1, [R0]            Load ASCII characters
LDRB    R2, [R0, #1]            into R1 and R2.
AND     R2, R2, #&F         Clear high-order 28 bits of R2.
ORR     R2, R2, R1, LSL #4  Shift contents of R1 left,
                                perform logical OR with
                                contents of R2, and place
                                result into R2.
STRB    R2, PACKED          Store packed BCD digits
                                into PACKED.
```

R2	00000000	00000000	00000000	00110010
F	00000000	00000000	00000000	00001111
R2&F	00000000	00000000	00000000	00000010
R1	00000000	00000000	00000000	00110110
R1<<4	00000000	00000000	00000011	01100000
R2|(R1<<4)	00000000	00000000	00000011	01100010

ASCII-Byte1	ASCII-Byte2	ASCII-Byte3	ASCII-Byte4
Loaded to R1	Loaded to R2		

Flags and Their Use

ARM ASSEMBLY LANGUAGE Fundamentals and Techniques, Ch. 7.2

CPSR: Current program status register

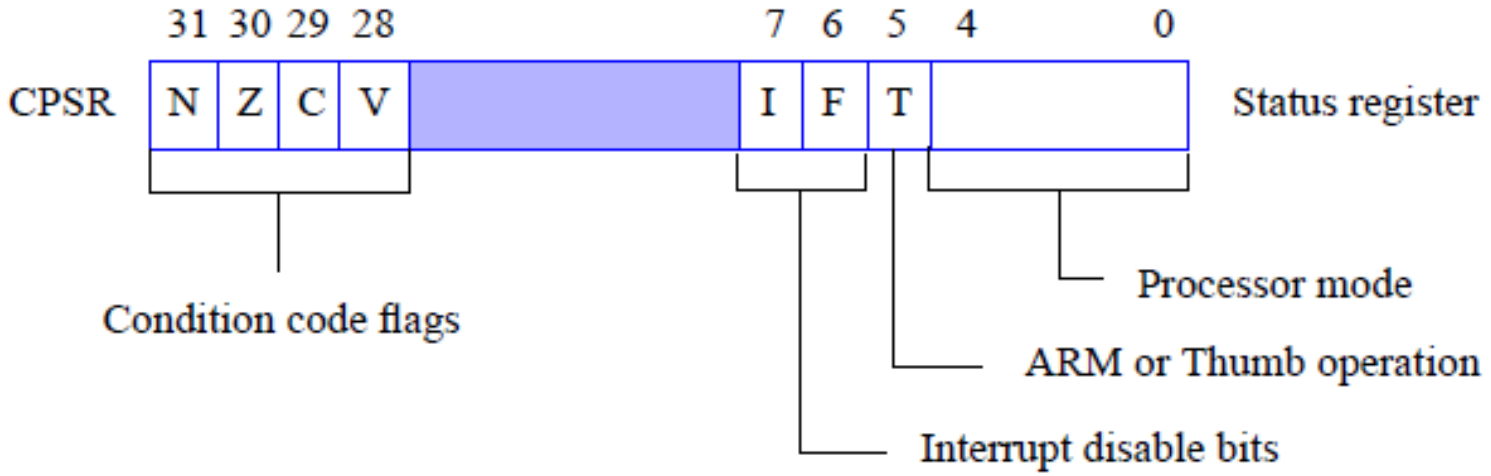


Figure 1. ARM register structure.

Condition code flags

- **N (negative)** Set to 1 if the result is negative;
  otherwise, cleared to 0
- **Z (zero)** Set to 1 if the result is 0;
  otherwise, cleared to 0
- **V (overflow)** Set to 1 if arithmetic overflow occurs;
  otherwise, cleared to 0
- **C (carry)** Set to 1 if a carry-out results from the operation;
  otherwise, cleared to 0

Why set or clear the flags?

The flags are set and cleared based on:

- Instructions that are specifically used for setting and clearing flags, such as `TST` or `CMP`
- Instructions that are told to set the flags by appending an "**S**" to the mnemonic. For example, EORS would perform an exclusive OR operation and set the flags afterward, since the S bit is set in the instruction. We can do this with all of the ALU instructions, so we control whether or not to update the flags
- A direct write to the Program Status Register, where you explicitly set or clear flags

The Z Flag: Zero

- the Z flag tells us is that the result of an operation produces zero.

The C Fla g

- The Carry flag is set if the result of an addition is greater than or equal to 2^{32}, if the result of a subtraction is positive, or as the result of an inline barrel shifter operation in a move or logical instruction.

```
LDR   r3, =0x7B000000
LDR   r7, =0xF0000000
ADDS  r4, r7, r3  ; value exceeds 32 bits, generates C out
```

The N Flag: Negative

- This flag is useful when checking for a negative result.
- A two's complement number is considered to be negative if the most significant bit is set.
- Be careful, you could easily have two perfectly good positive numbers add together to produce a value with the uppermost bit set.
- Example: Add "-1" and "-2".

```
  FFFFFFFF
+ FFFFFFFE
----------
  FFFFFFFD
```

```
MOV   r3, #-1
MOV   r4, #-2
ADDS  r3, r4, r3
```

N flag becomes 1.

Example: Add 2 positive numbers

```
  7B000000
+ 30000000
----------
  AB000000
```

N flag becomes 1.

The V Flag: Overflow

- Calculate the V flag as an exclusive OR of the carry bit going into the most significant bit of the result with the carry bit coming out of the most significant bit,
- Then the **V flag accurately indicates a signed overflow**. Overflow occurs if the result of an add, subtract, or compare is greater than or equal to 2^{31}, or less than -2^{31}.

```
  A1234567
+ B0000000
-----------
 151234567
```

We lost this bit

5: 0101

Result does not fit into 32 bits. We added two fairly large, negative numbers together, result should be negative, but we lost bit 33. The MSB of the 32-bit result is clear (notice the 5 in the most significant byte of the result).

```
  7B000000
+ 30000000
-----------
  AB000000
```

```
LDR   r3, =0x7B000000
LDR   r4, =0x30000000
ADDS  r5, r4, r3
```

The result fit into 32 bits. However, the result would be interpreted as a negative number when we started off adding two positive numbers, so is this an overflow case?
YES. Both the N and the V bits would be set in the CPSR.

Setting Condition Code Flags

- The Compare and Test instructions always update the condition code flags.
- The arithmetic, logic, and Move instructions affect the condition code flags **only if explicitly specified** to do so by a bit in the OP-code field.

This is indicated by appending the suffix **S**

```
ADDS R0, R1, R2      // sets the condition code flags
ADD R0, R1, R2       // does not set the condition code flags
```

Branch Instructions

MEF UNIVERSITY
Your Freedom in Learning

A.1.7 B

B (Branch) transfers program execution to the address specified by label.

Syntax

```
B{cond}{.W} label
```

where:

cond is an optional condition code. See Section 6.1.2.

label is a PC-relative expression.

.W is an optional instruction width specifier to force the use of a 32-bit instruction in Thumb.

Table D.2 Condition field encoding in ARM instructions.

Condition field $b_{31} \ldots b_{28}$	Condition suffix	Name	Condition code test
0 0 0 0	EQ	Equal (zero)	$Z = 1$
0 0 0 1	NE	Not equal (nonzero)	$Z = 0$
0 0 1 0	CS/HS	Carry set/Unsigned higher or same	$C = 1$
0 0 1 1	CC/LO	Carry clear/Unsigned lower	$C = 0$
0 1 0 0	MI	Minus (negative)	$N = 1$
0 1 0 1	PL	Plus (positive or zero)	$N = 0$
0 1 1 0	VS	Overflow	$V = 1$
0 1 1 1	VC	No overflow	$V = 0$
1 0 0 0	HI	Unsigned higher	$\overline{C} \vee Z = 0$
1 0 0 1	LS	Unsigned lower or same	$\overline{C} \vee Z = 1$
1 0 1 0	GE	Signed greater than or equal	$N \oplus V = 0$
1 0 1 1	LT	Signed less than	$N \oplus V = 1$
1 1 0 0	GT	Signed greater than	$Z \vee (N \oplus V) = 0$
1 1 0 1	LE	Signed less than or equal	$Z \vee (N \oplus V) = 1$
1 1 1 0	AL	Always	
1 1 1 1		not used	

Default

Branch examples

```
B LOC       // branch to the location in the instruction memory.
BEQ LOC     // branch if equal, Z = 1
BNE LOC     // branch is not equal, Z = 0
```

Branch offset generation

- Conditional branch instructions contain a 24-bit 2's-complement value that is used to generate a branch offset
- The value in the instruction is shifted left two bit positions then sign-extended to 32 bits to generate the offset. This offset is added to the updated contents of the PC to generate the branch target address.
- The appropriate 24-bit value in the instruction is computed by the assembler.

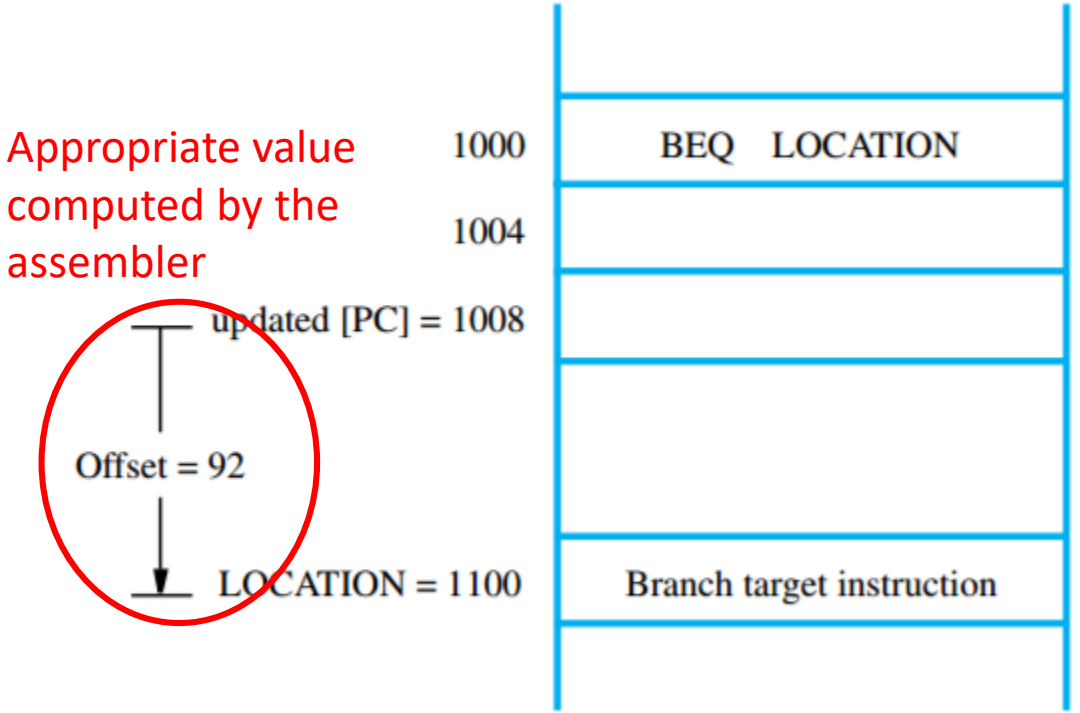


Figure D.6 Determination of the target address for a branch instruction.

Example - A Program for Adding Numbers

```
        LDR R1, N           // Load count into R1.
        LDR R2, =NUM1       // Load address NUM1 into R2.
        MOV R0, #0          // Clear accumulator R0.
LOOP    LDR R3, [R2], #4    // Load next number into R3.
        ADD R0, R0, R3      // Add number into R0.
        SUBS R1, R1, #1     // Decrement loop counter R1.
        BGT LOOP            // Branch back if not done.
        STR R0, SUM         // Store sum
```

COMPARISON INSTRUCTIONS

do nothing except set the condition codes: ***CMP, CMN, TST, TEQ***

CMP—Compare. CMP subtracts a register or an immediate value from a register value and updates the condition codes. You can use CMP to check the contents of a register for a particular value, such as at the beginning or end of a loop.

CMN—Compare negative. CMN adds a register or an immediate value to another register and updates the condition codes. CMN can also check register contents. This instruction is actually the inverse of CMP, and the assembler will replace a CMP instruction when appropriate.

Compare Instructions

CMP Rn, Rm	[Rn] – [Rm]
CMN Rn, Rm	[Rn] + [Rm]

- They set the condition code flags.
- The second operand can be shifted.
- Example:

```
CMP r0, #-20          // the assembler will instead generate:
CMN r0, #0x14
```

Compare

A.1.24 CMP

CMP (Compare) performs a comparison by subtracting the value of Operand2 from the value in Rn. The condition flags are changed, based on the result, but the result itself is discarded.

Syntax

```
CMP{cond} Rn, <Operand2>
```

where:

cond is an optional condition code. See Section 6.1.2.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See Section 6.2.1.

COMPARISON INSTRUCTIONS

do nothing except set the condition codes: ***CMP, CMN, TST, TEQ***

TST—Test. TST logically ANDs an arithmetic value with a register value and updates the condition codes without affecting the V flag. You can use TST to determine if many bits of a register are all clear or if at least one bit of a register is set.

TEQ—Test equivalence. TEQ logically exclusive ORs an arithmetic value with a register value and updates the condition codes without affecting the V flag. You can use TEQ to determine if two values are the same.

Test Instructions

- TST : Test, logical XOR
- TEQ : Test Equivalence, logical AND
- They do not store the result in a register!

		Rn ~=1	Rn=1
	Rn	11001010	00000001
	#1	00000001	00000001
TST	Rn ⊕ 1	11001011	00000000
TEQ	Rn && 1	00000000	00000001

Zero flag is asserted.

Zero flag is asserted.

Example - A Program for Adding Test Scores

```
        LDR R1, =WA1
        LDR R2, =WH1S1
        MOV R3, #0
        LDR R4, #10                 //10 students
LOOP    LDR R5, [R2], #4
        ADD R3, R3,R5               //HW1
        LDR R5, [R2], #4
        ADD R3, R3, R5               //HW2
        LDR R5, [R2], #4
        ADD R3, R3, R5               //HW3
        LDR R5, [R2], #4
        ADD R3, R3, R5               //HW4
        MOV R3, R3, LSR, #3          ///8
        LDR R5, [R2], #4
        ADD R3, R3, R5, LSR, #1          //0.5xFinal is added
        STR R3, [R1], #4                 // save the weighted average
        SUBS R4,R4, #1                  //decrement counter
        BGT LOOP                    //loop back if still have students
```

ADDRESS	Content
WA1	32 bit
WA2	32 bit
WA3	32 bit
WA4	32 bit
WA5	32 bit
WA6	32 bit
WA7	32 bit
WA8	32 bit
WA9	32 bit
WA10	32 bit

ADDRESS	Content
HW1S1	32 bit
HW2S1	32 bit
HW3S1	32 bit
HW4S1	32 bit
Final1	32 bit
HW1S2	32 bit
HW2S2	32 bit
HW3S2	32 bit
HW4S2	32 bit
Final2	32 bit
.....	32 bit
	32 bit
	32 bit
Final10	32 bit

Example - A Program for Adding Test Scores

```
        LDR R1, =WA1
        LDR R2, =WH1S1
        MOV R3, #0
        LDR R4, #10              //10 students
LOOP    LDM R2!, {R5,R6,R7,R8,R9}   //grades of 1 student
        ADD R10, R5,R6          //HW1+HW2
        ADD R11, R7,R8           //HW3 + HW4
        ADD R11, R10,R11         //sum(HW)
        MOV R11, R11, LSR, #3          ///8
        ADD R11, R11, R9, LSR, #1    //0.5xFinal is added
        STR R11, [R1], #4                  // save the weighted average
        SUBS R4,R4, #1                     //decrement counter
        BGT LOOP                 //loop back if still have students
```

ADDRESS	Content
WA1	32 bit
WA2	32 bit
WA3	32 bit
WA4	32 bit
WA5	32 bit
WA6	32 bit
WA7	32 bit
WA8	32 bit
WA9	32 bit
WA10	32 bit

ADDRESS	Content
HW1S1	32 bit
HW2S1	32 bit
HW3S1	32 bit
HW4S1	32 bit
Final1	32 bit
HW1S2	32 bit
HW2S2	32 bit
HW3S2	32 bit
HW4S2	32 bit
Final2	32 bit
.....	32 bit
	32 bit
	32 bit
Final10	32 bit