SPRING 2016

CMPE 364

Microprocessor Based Design

Dr. Mohamed Al-Meer

Control Flow Instructions

ARM Architecture

Lecture Expectations

- Expected to achieve:
  - Know about **Compare** and **Test** Instructions and their equivalents.
  - Know about **Conditional Branch** Instructions and their equivalents.
  - Know how to convert from **HL conditional execution** (IF-ELSE) and (SWITCH) Statements into ARM Assembly equivalents.
- Solved Book's Problems

Introduction

- Control Flow Instructions enables the processor to make decisions
- used to implement high level language construct like IF THEN ELSE and CASE SWITCH
- **EXAMPLE**
  - **IF R0 < 0 THEN R0:=0 - R0;**
  - **R0:=R0 + R1;**
- In Assembly

```
              CMP R0, #0
              BGE DONE
              RSB R0, R0, #0
DONE          ADD R0, R0, R1
```

Compare and Test Instructions

- CMP Instruction
- Compares 2 values and update the condition code in CPSR based on the results
- Syntax

  CMP SRC1, SRC2
- **SRC1** must be a **register**
- SRC2 can be a register, an immediate, or a shifted/rotated register value
- Based on difference of SRC1 – SRC2 condition flags are set

CMP Instruction

- Conditions codes will be :
  - Zero (Z) set if the difference is zero
  - Negative (N) set if difference is negative
  - Carry (C) set if there is a carry from subtraction
  - Overflow (V) set if difference overflows 32-bit value
- EXAMPLE
  - Which condition codes in CPSR are set after the following instruction
  - **CMP R1, R3**
  - If R1 = 000000A4, R3 = 00000006?
- SOLUTION
  - Difference = 0000009E, Z=0, N=0, C=1, V=0.
  - Get 2’s complement of R3 then add to R1 and watch C and V flags.

CMP Instruction

```
 0000 0000 0000 0000 0000 0000 1010 0100 +        (R1)
 1111  1111  1111 1111  1111  1111 1 111 1010      (-R3)
 ----------------------------------------------------------
10000 0000 0000 0000 0000 0000 1001 1110
^ Carry Out
```

CMP Instruction

- **EXAMPLE**
  - Determine values of CPSR, R0, R5 after the following instruction
  - **CMP R4, R0**
  - If R0 = FFFFFFFE (-2), R4 = 00000C00, CPSR = 60000010?
- **SOLUTION**
  - Difference = 00000C02
  - Z=0, N=0, V=0, C=0, CPSR = 00000010

```
 0000 0000 0000 0000 0000 1100 0000 0000          (R4)
 0000 0000 0000 0000 0000 0000 0000 0010    +     (-R0)
 -----------------------------------------------------------------------
00000 0000 0000 0000 0000 1100 0000 0010 = 00000C02
^--- Carry Out
```

CMN Instruction

- The Compare Negative Instruction
- **Compares an arithmetic value with a negative of a second arithmetic value**
- Condition codes in CPSR are updated
- Syntax

  CMN SRC1, SRC2

The same conditions and restrictions of CMP applies here.

CMN Instruction

- **EXAMPLE**
  - What is the value stored in condition codes after the following instruction has completed?
  - **CMN R0, R6**
  - R0 = 000E804. R6 = 0000006B
- **SOLUTION**
  - **R0 and R6 are added together !!**

```
  0000 0000 0000 1110 1000 0000 0100
  0000 0000 0000 0000 0000 0110 1011  +
  -------------------------------------------------
00000 0000 0000 1110 1000 0110 1111
^---- Carry Out
```

CMN Instruction

  - ZZ= 0, N=0, C=0, V=0
- EXAMPLE
  - Consider the following instruction. Determine the value of the flags?
  - **CMN R8, R5**
  - R5 = FFFFFFC4, R8 = 0000003C
- SOLUTION
  - Do it yourself?

TEQ Instruction

- **TEQ** = **Test Equivalence instruction**
- **Compares** 2 arithmetic values SRC1 and SRC2, by taking **bitwise exclusive OR (EOR).** And correspondingly setting the flags in CPSR register
- **SRC1** must be a **register**
- **SRC2** can be a **register**, an **immediate**, or a **shifted/rotated value of a register**
- Overflow flag (V) will not change
- **Zero flag** (Z) can tell if the 2 values are the same or not

TEQ Instruction

- EXAMPLE
  - **TEQ R12, R9**
  - R9 = 800C2016, R12 = 9E0A6A92, CPSR = 30000013
- SOLUTION
  - 1001 1110 0000 1010 0110 1010 1001 0010
  - 1000 0000 0000 1100 0010 0000 0001 0110   XOR
  - ---
  - **0001 1110 0000 0110 0100 1010 1000 0100**
  - **Z=0**, N=0, V=unchanged, C=1 unchanged.
  - CPSR Unchanged.

TST Instruction

- Test Instruction
- **Compare** 2 arithmetic values by taking **bitwise AND of the two operands** and sets the condition codes in CPSR register only
- Used to determine if a **specific subset occurs within a bigger set**
- **Z** and **N** flags will be affected accordingly, and V flag is not affected
- **C flag** is set if the second value is processed by the **barrel shifter**.
- **SRC1** must be a **register**
- SRC2 could be a register, an immediate or a shifted/rotated value in a register.

TST Instruction

- Syntax

  TST SRC1, SRC2
- **EXAMPLE**
  - **TST R4, R9**
  - R4 = 709BD531, R9 = 601AD531, CPSR = E0000012?
- **SOLUTION**
  - R4 and R9 are **AND**ed together.
  - 0111 0000 1001 1011 1101 0101 0011 0001
  - 0110 0000 0001 1010 1101 0101 0011 0001 AND
  - ---
  - 0110 0000 0001 1010 1101 0101 0011 0001 **Z=?, N=?**

TST Instruction

- Z=0, N=0, V Unchanged, C Unchanged.
- **EXAMPLE**
  - Write an instruction to determine if **bit 27** of R8 is **set or not**?
- **SOLUTION**
  - **TST R8, #0x08000000**
  - Second source is a Mask
  - 1 in bit position 27 will be copied to result if ANDed with 1.
  - **Z flag will take the Inverse bit position 27 value**.
    - Bit 27 = 1 >> ZF = 0
    - Bit 27 = 0 >> ZF = 1

Branch Instructions

- Branch instruction allows program to execute in non sequential manner
- Conditional Branch allows ARM specifies the condition under which the program will branch.
- **Conditions** are evaluated using **condition codes** in **CPSR**
- Preceded instruction (before conditional branch) must modify the code flags in CPSR
- Next table in nest slide shows all the usage of conditional branches in the ARM

Branch Instructions

- See next program

```
Loop LDR  R1, (R4)
     ADD  R0, R0, R1
     CMP  R1, R5
     BNE  Loop
```

How many Bits?

- Note the branch target and the label “Loop”
- Label could be away from branch instructions +-32 MB in backward or forward.

Branch Instructions

- See next program

```
     CMP  R4, R8
     BLE  Next
     ADD  R0, R4, R8
Next SUB  R0, R0, #17
```

Which One???

- Assume R4 = 00000051, R8 = 00000AC9.
    - **Branch will be taken if R4 <= R8**
    - **If R4 > R8 then branch will not be taken.**
- R0 = **??????**

Branch Instructions

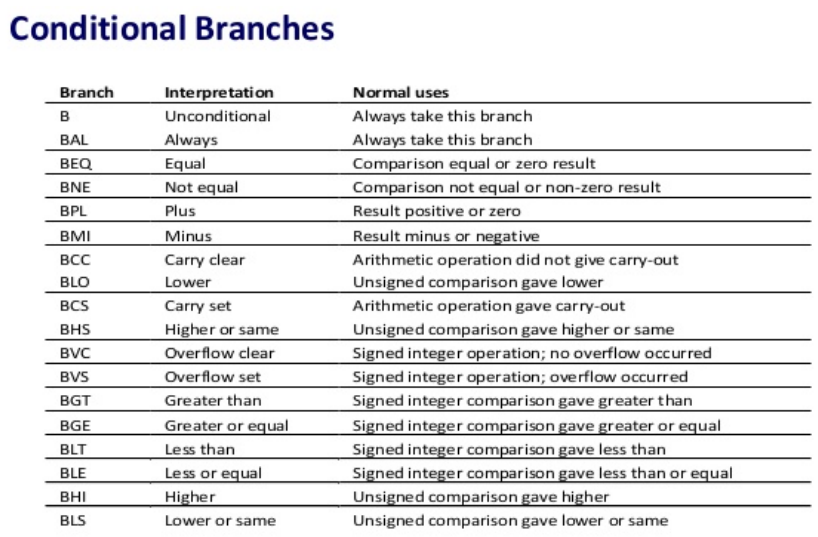
How controls Loop Number?

- See next code
- How many times the loop body will be executed?

```
        MOV R1, #5
Process ADD  R0, R0, R1
        SUB  R1, R1, #1
        CMP  R1, #0
        BGE  Process
```

- Answer: **(4, 5, 6, 7, 8) ??**

Branch Instructions



Conditional Branches

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

Branch Instructions

- See the next Example

Write an ARM Assembly code to: complement bit **5** of R4 if bit **9** of R4 = 1. Clear bit **6** of R4?

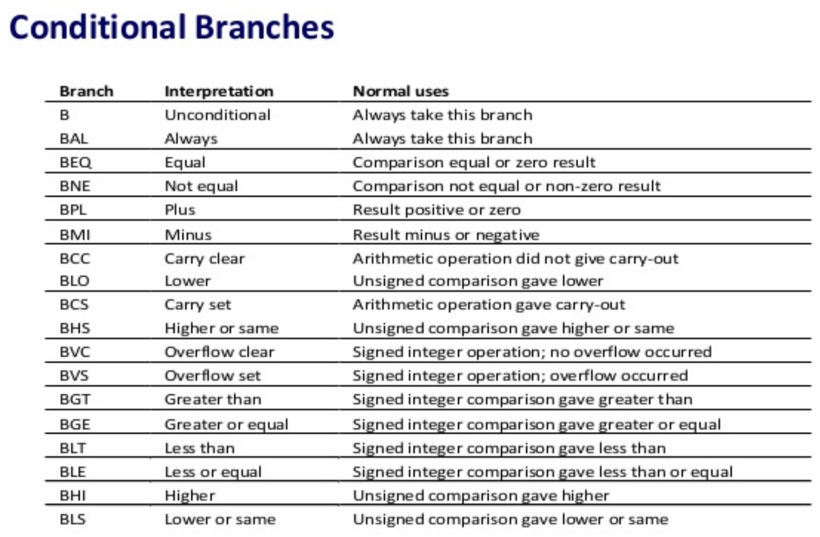
```
          TST  R4, #0x200
          BEQ  Continue
          EOR  R4, R4, #0x20
Continue  BIC  R4,R4, 0x40
```

Branch Instructions

- Implement the next IF THEN ELSE statement into Assembly?
  - **IF R2 < R7 THEN R7 = R7 +3**
  - **ELSE R2 = R2 + 3**
- Solution

```
                CMP  R2, R7
                BGE  ELSE
                ADD  R7, R7, #3
                BAL  DONE       ; B    DONE
ELSE            ADD  R2, R2, #3
DONE
```

Branch Instructions



Conditional Branches

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

Branch Instructions cont.

```
Syntax: B{<cond>} label
        BL{<cond>} label
        BX{<cond>} Rm
        BLX{<cond>} label | Rm
```

B	branch	*pc* = *label*
BL	branch with link	*pc* = *label* *lr* = address of the next instruction after the BL
BX	branch exchange	*pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1
BLX	branch exchange with link	*pc* = *label*, *T* = 1 *pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1 *lr* = address of the next instruction after the BLX

Branch Instructions (BL)

- Branch and Link
- Used to call subroutines and come back
- **R14** or ***lr*** will save PC +4
- When subroutine finishes the user executes
  - MOV PC, lr
- **Example**

```
BL    sort
```

Branch Instructions (BL)

- This example shows a simple fragment of code that branches to a subroutine using the **BL** instruction. To return from a subroutine, you copy the link register to the pc.

```
        BL      subroutine      ; branch to subroutine
        CMP     r1, #5          ; compare r1 with 5
        MOVEQ   r1, #0          ; if (r1==5) then r1 = 0
        :
subroutine
        <subroutine code>
        MOV     pc, lr          ; return by moving pc = lr
```