COMP 122/L Lecture 9

Mahdi Ebrahimi

Slides adapted from Dr. Kyle Dewey

Outline

- The compare (`cmp`) instruction
- Conditionally-executed instructions
- Translating simple `if` statements

The compare (`cmp`) instruction

Compare (`cmp`)

Subtracts two given operands, discarding the result. However, the status bits (e.g., carry, zero, etc.) get set.

Syntax

CMP Rn, Operand2

where:

Rn is the ARM register holding the first operand.

Operand2 is a flexible second operand.

Operation

These instructions compare the value in a register with Operand2. They update the condition flags on the result, but do not place the result in any register.

The CMP instruction subtracts the value of Operand2 from the value in Rn. This is the same as a SUBS instruction, except that the result is discarded.

Compare (`cmp`)

Subtracts two given operands, discarding the result. However, the status bits (e.g., carry, zero, etc.) get set.

```
mov r0, #5
cmp r0, #5
```

Compare (`cmp`)

Subtracts two given operands, discarding the result. However, the status bits (e.g., carry, zero, etc.) get set.

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

Compare (`cmp`)

Subtracts two given operands, discarding the result. However, the status bits (e.g., carry, zero, etc.) get set.

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

```
mov r0, #5
cmp r0, #20
```

Compare (`cmp`)

Subtracts two given operands, discarding the result. However, the status bits (e.g., carry, zero, etc.) get set.

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

```
mov r0, #5
cmp r0, #20
```

Sets negative bit/flag
(result is negative)

Significance

Status bits say something about
the result of arithmetic comparisons

Significance

Status bits say something about the result of arithmetic comparisons

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

Significance

Status bits say something about
the result of arithmetic comparisons

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

Operands must have been equal.

Significance

Status bits say something about
the result of arithmetic comparisons

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

Operands must have been equal.

```
mov r0, #5
cmp r0, #20
```

Sets negative bit/flag
(result is negative)

Significance

Status bits say something about
the result of arithmetic comparisons

```
mov r0, #5
cmp r0, #5
```

Sets zero bit/flag
(result is zero)

Operands must have been equal.

```
mov r0, #5
cmp r0, #20
```

Sets negative bit/flag
(result is negative)

First operand must be < second.

Conditionally-executed instructions

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
movmi r0, #42
```

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
movmi r0, #42
```

move if the negative bit is set

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
movmi r0, #42
```

move if the negative bit is set

```
movpl r1, #23
```

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
movmi r0, #42
```

move if the negative bit is set

```
movpl r1, #23
```

move if the negative bit is **not** set

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
moveq r0, #42
```

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
moveq r0, #42
```

move if the zero bit is set

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

```
moveq r0, #42
```

move if the zero bit is set

```
movne r0, #42
```

Conditionally-Executed Instructions

ARM allows for instructions to be *conditionally* executed, depending on the values of the status bits.

`moveq r0, #42`

move if the zero bit is set

`movne r0, #42`

move if the zero bit is **not** set

Basic data processing instructions

MOV	Move a 32-bit value	MOV Rd,n	Rd = n
MVN	Move negated (logical NOT) 32-bit value	MVN Rd,n	Rd = ~n
ADD	Add two 32-bit values	ADD Rd,Rn,n	Rd = Rn+n
ADC	Add two 32-bit values and carry	ADC Rd,Rn,n	Rd = Rn+n+C
SUB	Subtract two 32-bit values	SUB Rd,Rn,n	Rd = Rn-n
SBC	Subtract with carry of two 32-bit values	SBC Rd,Rn,n	Rd = Rn-n+C-1
RSB	Reverse subtract of two 32-bit values	RSB Rd,Rn,n	Rd = n-Rn
RSC	Reverse subtract with carry of two 32-bit values	RSC Rd,Rn,n	Rd = n-Rn+C-1
AND	Bitwise AND of two 32-bit values	AND Rd,Rn,n	Rd = Rn AND n
ORR	Bitwise OR of two 32-bit values	ORR Rd,Rn,n	Rd = Rn OR n
EOR	Exclusive OR of two 32-bit values	EOR Rd,Rn,n	Rd = Rn XOR n
BIC	Bit clear. Every '1' in second operand clears corresponding bit of first operand	BIC Rd,Rn,n	Rd = Rn AND (NOT n)
CMP	Compare	CMP Rd,n	Rd-n & change flags only
CMN	Compare Negative	CMN Rd,n	Rd+n & change flags only
TST	Test for a bit in a 32-bit value	TST Rd,n	Rd AND n, change flags
TEQ	Test for equality	TEQ Rd,n	Rd XOR n, change flags

MUL	Multiply two 32-bit values	MUL Rd,Rm,Rs	Rd = Rm*Rs
MLA	Multiple and accumulate	MLA Rd,Rm,Rs,Rn	Rd = (Rm*Rs)+Rn

- Features of Conditional Execution instructions

Improves execution speed and offers high code density

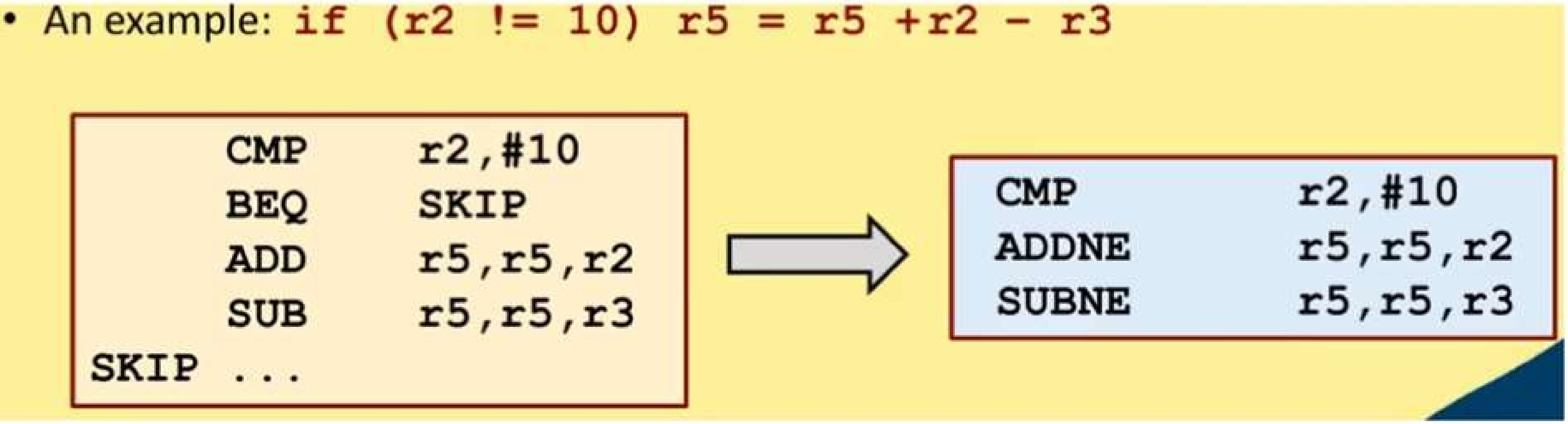
Illustration:

'C' Program fragment	ARM program using branching instructions	ARM program using conditional instructions
`if (r0==0)`	`CMP r0,#0`	`CMP r0,#0`
`{`	`BNE else`	`ADDEQ r1,r1,#1`
`r1=r1+1;`	`ADD r1,r1,#1`	`ADDNE r2,r2,#1`
`}`	`B end`	
`else`	`else ADD r2,r2,#1`	
`{`	`end ---`	
`r2=r2+1;`		
`}`	Instructions - 5	Instructions - 3
	Memory space - 20 bytes	Memory space - 12 bytes
	No. of cycles - 5 or 6	No. of cycles - 3

Example:
`conditional_execution.s`

Translating simple `if` statements

Example 1

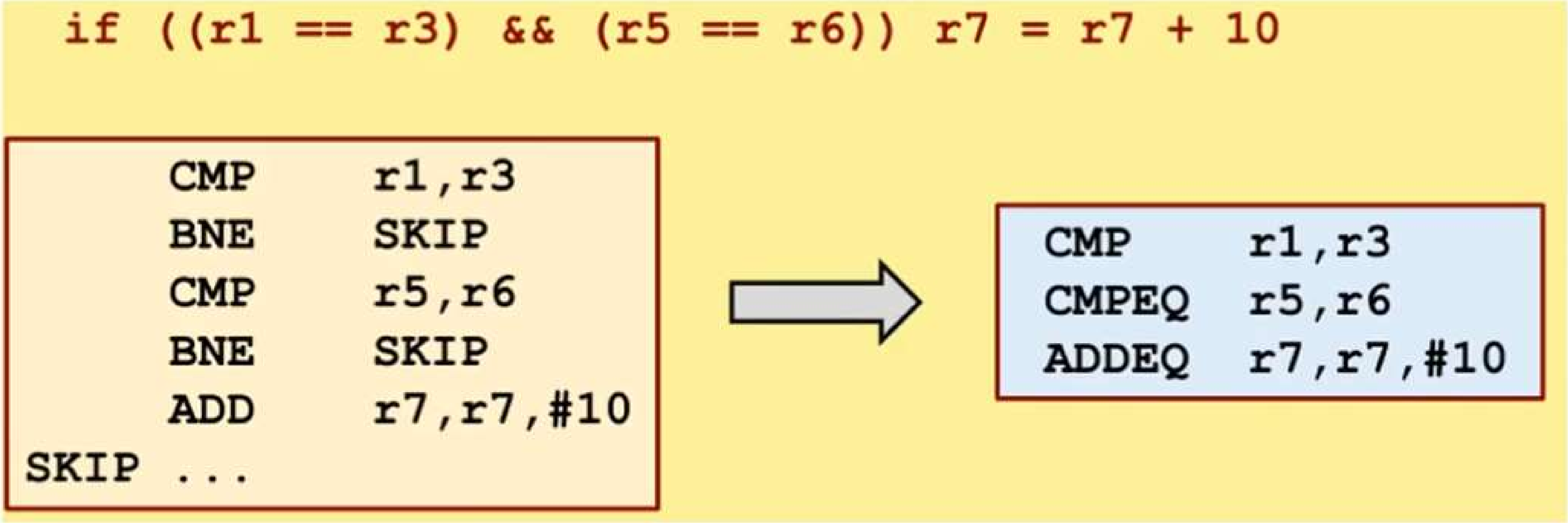


- An example: `if (r2 != 10) r5 = r5 +r2 - r3`

```
        CMP     r2,#10
        BEQ     SKIP
        ADD     r5,r5,r2
        SUB     r5,r5,r3
SKIP    ...
```

```
CMP             r2,#10
ADDNE           r5,r5,r2
SUBNE           r5,r5,r3
```

Example 2



```
if ((r1 == r3) && (r5 == r6)) r7 = r7 + 10
```

```
      CMP     r1,r3
      BNE     SKIP
      CMP     r5,r6
      BNE     SKIP
      ADD     r7,r7,#10
SKIP ...
```

```
CMP     r1,r3
CMPEQ   r5,r6
ADDEQ   r7,r7,#10
```

Translating `if`

- Simple `if`s can be translated with conditionally-executed instructions
- Example:
  - `AbsoluteValue.java`
  - `absolute_value.s`