
ELEC2041 Microprocessors and Interfacing

Lectures 14: Making Decisions in C/Assembly Language - II

<http://webct.edtec.unsw.edu.au/>

March 2006

Saeid Nooshabadi

saeid@unsw.edu.au

ELEC2041 lec14-decision-II.1

Saeid Nooshabadi

Overview

- Compare instruction mechanism
- Flag setting Instructions
- Conditional Instructions
- Conclusion

ELEC2041 lec14-decision-II.2

Saeid Nooshabadi

Review (#1/2)

- HLL decisions (if, case) and loops (while, for) use same assembly instructions

- Compare instruction: `cmp` in ARM
- Conditional branches: `beq`, `bne`, `bgt`, `blt`, etc in ARM
- Unconditional branches: `b`, and `mov pc, rn` in ARM
- Switch/Case: chained if-else or jump table + `ldr pc, []`

• `ldr pc, []` is VERY POWERFUL!

ELEC2041 lec14-decision-II.3

Saeid Nooshabadi

Review (#2/2)

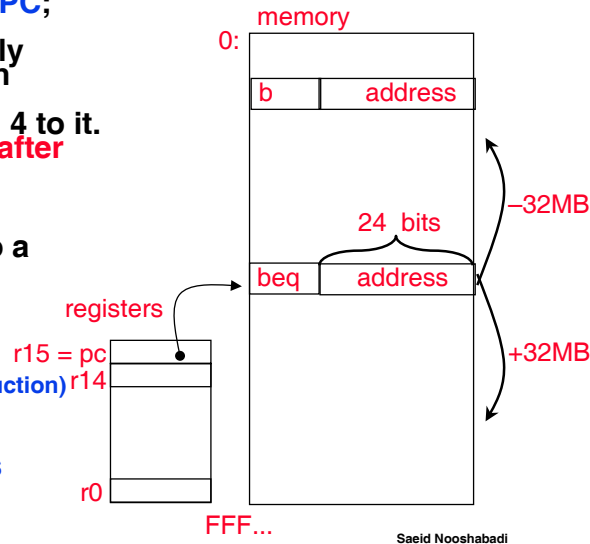
- Some Branch Conditions after `cmp` instruction:
 - `b` Unconditional
 - `bal` Branch Always
 - `beq` Branch Equal
 - `bne` Branch Not Equal
 - `blt` Branch Less Than
 - `ble` Branch Less Than or Equal
 - `bgt` Branch Greater Than
 - `bge` Branch Greater Than or Equal
- Full Table Page 64 Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6.

ELEC2041 lec14-decision-II.4

Saeid Nooshabadi

Review: Branches: PC-relative addressing

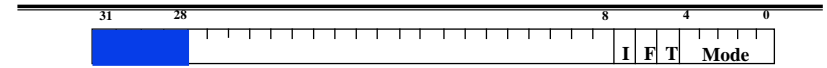
- Recall register **r15** in the machine also called **PC**;
- points to the currently executing instruction
- Most instruction add 4 to it. (pc increments by 4 after execution of most instructions)
- Branch changes it to a specific value
- Branch adds to it
 - 24-bit signed value (contained in the instruction) r14
 - Shifted left by 2 bits
- Labels => addresses



ELEC2041 lec14-decision-II.5

Saeid Nooshabadi

Review: Status Flags in Program Status Register CPSR



Copies of the ALU status flags (latched for some instructions).

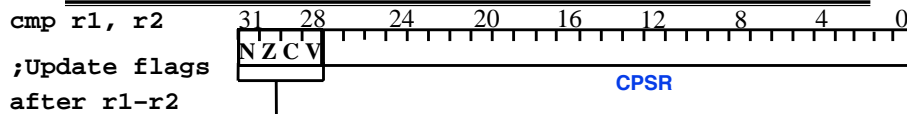
`cmp r1, r2 ; update Flags after r1-r2`

Flags	After <code>cmp</code> Instruction
Negative (N='1')	Bit 31 of the result has been set. Indicates a negative number in signed operations
Zero (Z='1')	Result of operation was zero.
Carry (C='1')	Result was greater than 32 bits (for unsigned arithmetic, bit 31 is the magnitude bit , and not the sign bit).
oVerflow (V='1')	Result was greater than 31 bits (for signed arithmetic bit 31 is the sign bit and not the magnitude bit). Indicates a possible corruption of the sign bit in signed numbers (carry into the msb \neq carry out of msb)

ELEC2041 lec14-decision-II.6

Saeid Nooshabadi

Cmp Instructions and CPSR Flags (#1/3)

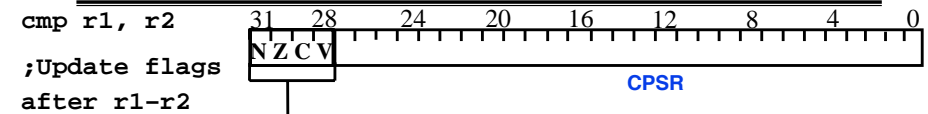


- x1xx** = Z set (equal)(eq)
- x0xx** = Z clear (not equal)(ne)
- xx1x** = C set (unsigned higher or same) (hs/cs)
- xx0x OR x1xx** = C clear or Z set (unsigned lower or same)(ls)
- xx0x** = C clear (unsigned lower) (lo/cc)
- x01x** = C set and Z clear (unsigned higher)(hi)
- 1xxx** = N set (negative) (mi)
- 0xxx** = N clear (positive or zero) (pl)

ELEC2041 lec14-decision-II.7

Saeid Nooshabadi

Cmp Instructions and CPSR Flags (#2/3)

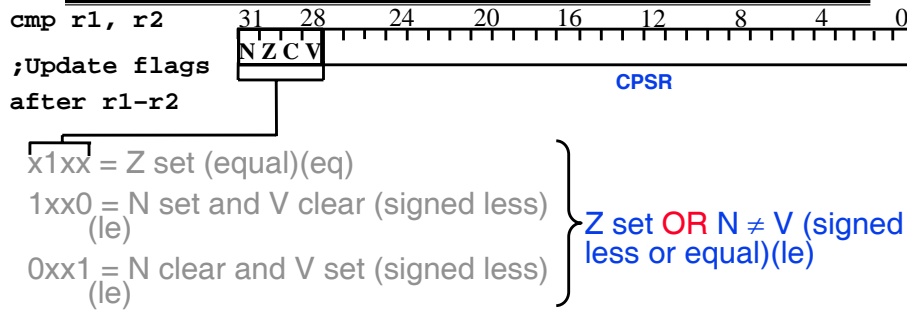


- xxx1** = V set (signed overflow)(vs)
 - xxx0** = V clear (signed no overflow) (vc)
 - 1xx1** = N set and V set (signed greater or equal) (ge)
 - 0xx0** = N clear and V clear (signed greater or equal)(ge)
 - 1xx0** = N set and V clear (signed less than) (lt)
 - 0xx1** = N clear and V set (signed less than) (lt)
 - 10x1** = Z clear, and N set and V set (signed greater)(gt)
 - 00x0** = Z clear, and N clear and V clear (signed greater) (gt)
- Additional relationships shown:
- N = V** (signed greater or equal) (ge)
 - N \neq V** (signed less than) (lt)
 - Z clear AND (N = V)** (signed greater than) (gt)

ELEC2041 lec14-decision-II.8

Saeid Nooshabadi

cmp Instructions and CPSR Flags (#3/3)



Example

Assuming $r1 = 0x7fffffff$, and $r2 = 0xffffffff$

What the CPSR flags would be after the instruction; `cmp r1, r2`

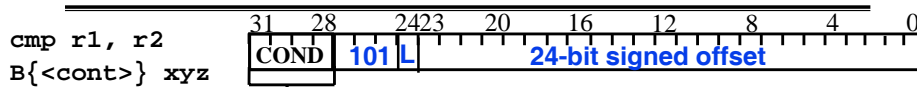
Answer: $N=1, Z=0, C=0, V=1$

$r1 = 2147483647$
 $r2 = (4294967295) \text{ or } (-1)$
 (Unsigned) (signed)

$r1 - r2 = 0x7fff\ ffff - 0xffff\ ffff$
 $r1 - r2 = 0x7fff\ ffff + 0xffff\ ffff$
 $r1 - r2 = 0x7fff\ ffff + 0x0000\ 0001$
 $0x8000\ 0000$ (carry into msb \neq carry out of msb)
0 (carry out)

C clear \rightarrow (unsigned lower) (lo/cc)
 Z clear AND ($N=V$) \rightarrow (signed greater than) (gt)
 $2147483647 > -1$
 2's complement of $0xffff\ ffff$

Branch Instruction Conditional Execution Field



0000 = EQ - Z set (equal)
 0001 = NE - Z clear (not equal)
 0010 = HS / CS - C set (unsigned higher or same)
 0011 = LO / CC - C clear (unsigned lower)
 0100 = MI - N set (negative)
 0101 = PL - N clear (positive or zero)
 0110 = VS - V set (overflow)
 0111 = VC - V clear (no overflow)
 1000 = HI - C set and Z clear (unsigned higher)
 1001 = LS - C clear or Z set (unsigned lower or same)
 1010 = GE - N set and V set, or N clear and V clear (signed $>$ or $=$)
 1011 = LT - N set and V clear, or N clear and V set (signed $<$)
 1100 = GT - Z clear, and either N set and V set, or N clear and V clear (signed $>$)
 1101 = LE - Z set, or N set and V clear, or N clear and V set (signed $<$, or $=$)
 1110 = AL - always
 1111 = NV - reserved

Flag Setting Instructions

- Compare
`cmp r1, r2 ; Update flags after r1 - r2`
- Compare Negated
`cmn r1, r2 ; Update flags after r1 + r2`
- Test
`tst r1, r2 ; Update flags after r1 AND r2`
- Test Equivalence
`teq r1, r2 ; Update flags after r1 EOR r2`

These instructions DO NOT save results; Only UPDATE CPSR Flags

Flag Setting Instructions Example

Assuming $r1 = 0x7fffffff$, and $r2 = 0xffffffff$

What the CPSR flags would be after the instructions

```
cmp r1, r2, lsl #2
cmn r1, r2, lsl #2
tst r1, r2, lsr #1
teq r1, r2, lsr #1
```

Flag Setting Instructions Example Solution (#1/4)

```
cmp r1, r2, lsl #2
```

Answer:

```
r1 = 0x7fffffff = 2147483647
r2 lsl #2 = 0xffffffffc
           = 4294967292 (unsigned)
           = (-4) (signed)
```

```
r1 - (r2, lsl #2) =
0x7fff ffff +
0x0000 0004 2's complement of 0xffff fffc
-----
0x8000 0003 (carry into msb ≠ carry out of msb)
0 (carry out)
```

$N=1, Z=0, C=0, V=1$

C clear → (unsigned lower) (lo/cc)

$2147483647 < 4294967292$

Z clear AND ($N=V$) → (signed greater than) (gt)

$2147483647 > -4$

Flag Setting Instructions Example Solution (#2/4)

```
cmn r1, r2, lsl #2
```

Answer:

```
r1 = 0x7fffffff = 2147483647
r2 lsl #2 = 0xffffffffc
           = 4294967292 (unsigned)
           = (-4) (signed)
```

```
r1 + r2, lsl #2 = r1 - (-(r2 << 2)) (comparing r1 and -r2)
```

```
0x7fff ffff +
```

```
0xffff fffc
```

```
-----
0x7fff fffb (carry into msb = carry out of msb)
```

1 (carry out)

$N=0, Z=0, C=1, V=0$

C set → (unsigned higher or same) (hs/cc).

C set here really means there was an unsigned addition overflow

Z clear AND ($N=V$) → (signed greater than) (gt)

$2147483647 > -(-4) = 4$

Flag Setting Instructions Example Solution (#3/4)

```
tst r1, r2, lsr #1
```

Answer:

```
r1 = 0x7fffffff
R2, lsr #1 = 0x7fffffff
```



```
r1 and r2 lsr # 1 =
```

```
0x7fff ffff and
```

```
0x7fff ffff
```

```
-----
0x7fff ffff
```

$N=0, Z=0, C=1, V=0$

N clear → (Positive or Zero) (pl)

Z clear → (Not Equal) (ne). It really means ANDing of $r1$ and $r2$ does not make all bits 0, ie $r1 \text{ AND } r2 \neq 0$

Flag Setting Instructions Example Solution (#4/4)

```
teq r1, r2, lsr #1
```

Answer:

```
r1 = 0x7fffffff  
R2, lsr #1 = 0x7fffffff
```



```
r1 eor r2 lsl # 1 =  
0x7fff ffff xor  
0x7fff ffff  
-----  
0x0000 0000
```

N = 0, Z = 1, C = 1, V = 0

N clear → (Positive or Zero) (pl)
Z set → (Equal) It means $r1 = r2$

Updating CPSR Flags with Data Processing Instructions

◦ By default, data processing operations do not affect the condition flags

```
- add r0,r1,r2 ; r0 = r1 + r2  
; ... and DO not set  
; flags
```

◦ To cause the condition flags to be updated, the “S” bit of the instruction needs to be set by postfixing the instruction with an “S”.

• For example to add two numbers and set the condition flags:

```
- adds r0,r1,r2; r0 = r1 + r2  
; ... and DO set flags
```

Updating Flags Example

◦ Compile this C code into ARM:

```
sum = 0;  
for (i=0;i<10;i=i+1)  
    sum = sum + A[i];
```

• sum:v1, i:v2, base address of A:v3

Updating Flags Example Solution Ver 1

```
sum = 0;  
C for (i=0;i<10;i=i+1)  
    sum = sum + A[i];
```

• sum:v1, i:v2, base address of A:v3

```
ARM  
mov v1, #0  
mov v2, #0  
Loop: ldr a1,[v3,v2,ls1 #2] ; a1=A[i]  
add v1, v1, a1 ;sum = sum+A [i]  
add v2, v2, #1 ;increment i  
cmp v2, #10 ; Check(i<10)  
bne Loop ; goto loop
```

Updating Flags Example Solution Ver 2

```
C sum = 0;
  for (i=0;i<10;i=i+1)
    sum = sum + A[i];
```

• sum:v1, i:v2, base address of A:v3

```
A Loop: mov v1, #0
R       mov v2, #9           ; start with i = 9
M       ldr a1,[v3,v2,ls1 #2] ; a1=A[i]
        add v1, v1, a1       ;sum = sum+A [i]
        sub v2, v2, #1       ;decrement i
        cmp v2, #0           ; Check(i<0)
        bge Loop            ; goto loop
```

Updating Flags Example Solution Ver 3

```
C sum = 0;
  for (i=0;i<10;i=i+1)
    sum = sum + A[i];
```

• sum:v1, i:v2, base address of A:v3

```
A Loop: mov v1, #0
R       mov v2, #9           ; start with i = 9
M       ldr a1,[v3,v2,ls1 #2] ; a1=A[i]
        add v1, v1, a1       ; sum = sum+A [i]
        subs v2, v2, #1      ; decrement i
                                   ; update flags
        bge Loop            ; goto loop
```

ELEC2041 Reading Materials (Week #5)

- Week #5: Steve Furber: ARM System On-Chip; 2nd Ed, Addison-Wesley, 2000, ISBN: 0-201-67519-6. We use [chapters 3 and 5](#)
- ARM Architecture Reference Manual –On CD ROM

Conditional Execution

- Recall Conditional Branch Instruction: beq, bne, bgt, bge, blt, ble, bhi, bhs, blo, bls, etc
 - Almost all processors only allow branch instructions to be executed conditionally.
- However by reusing the condition evaluation hardware, ARM effectively increases number of instructions.
 - All instructions contain a condition field which determines whether the CPU will execute them.
- This removes the need for many branches, Allows very dense in-line code, without branches.
- Example:

```
subs v2, v2, #1           ; Update flags
addeq v3, v3, #2         ; add if EQ (Z = 1)
```

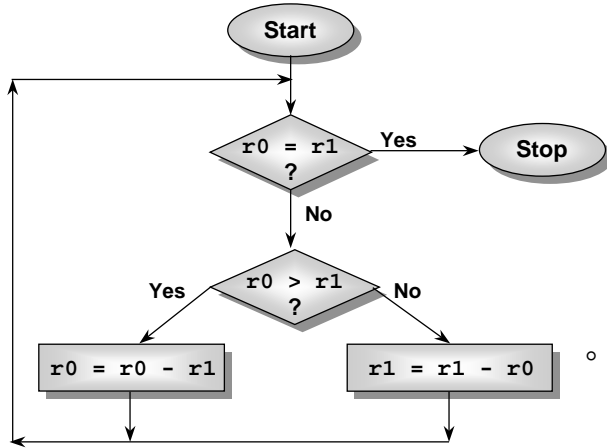
Conditional Code Example

- Convert the GCD algorithm given in this flowchart into

1) "Normal" assembler, where only branches can be conditional.

2) ARM assembler, where all instructions are conditional, thus improving code density.

- The only instructions you need are `cmp`, `b` and `sub`.



ELEC2041 lec14-decision-II.25

Saeid Nooshabadi

Conditional Code Example (Solution)

- "Normal" Assembler

```

gcd    cmp    r0, r1    ;reached the end?
        beq    stop
        blt    less    ;if r0 > r1
        sub    r0, r0, r1 ;subtract r1 from r0
        bal    gcd
less   sub    r1, r1, r0 ;subtract r0 from r1
        bal    gcd
stop
    
```

- ARM Conditional Assembler

```

gcd    cmp    r0, r1    ;if r0 > r1
        subgt  r0, r0, r1 ;subtract r1 from r0
        sublt  r1, r1, r0 ;else subtract r0
                        ; from r1
        bne    gcd      ;reached the end?
    
```

ELEC2041 lec14-decision-II.26

Saeid Nooshabadi

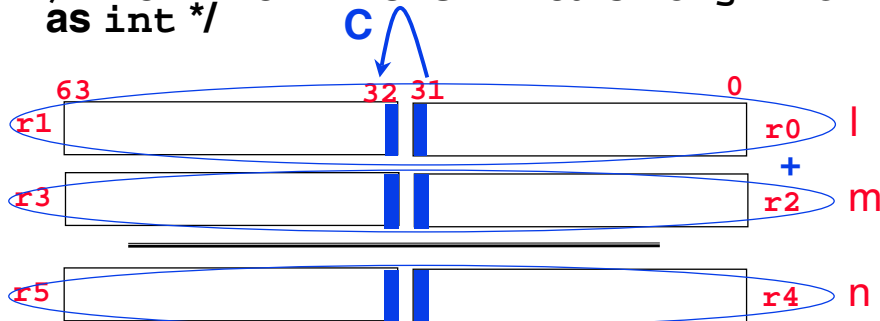
Long Integer Addition Example

- long int = 64 bits

long int l, m, n;

n = l + m;

*/*Won't work in C. Still treats long int as int*/*



ELEC2041 lec14-decision-II.27

Saeid Nooshabadi

Long Integer Addition Example (Analysis)

- We need to do the addition in two steps

```

struct int64 {int lo;
              int hi;
    
```

```

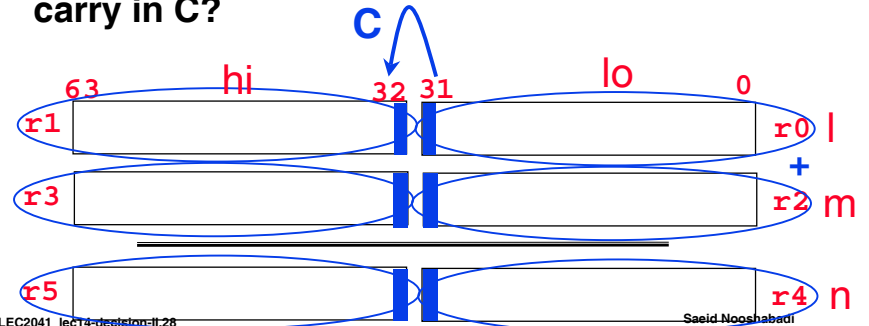
}l, m, n;
    
```

```

n.lo = m.lo + l.lo;
    
```

```

n.hi = m.hi + l.hi + C; How to check on
carry in C?
    
```



ELEC2041 lec14-decision-II.28

Saeid Nooshabadi

Long Integer Addition Example in C (GCC)

◦ long long type extension in gcc

```
long long ad_64(long long l, long long m){  
    long long n;  
    n = m + l;  
    return n;          Long long type is 64 bits  
}
```

◦ ARM Compiled Code

```
ad_64:  
/* l.lo, l.hi, m.lo and m.hi are passed in r0-  
r1*/  
    adds    r0, r0, r2 ; n.lo = l.lo + m.lo  
    adc     r1, r3, r1 ; n.hi = l.hi + m.hi + C  
    mov     pc, lr
```

“And in Conclusion ...”

- Flag Setting Instructions: `cmp`, `cmn`, `tst`, `teq` in ARM
- Data Processing Instructions with Flag setting Feature: `adds`, `subs`, `ands`, in ARM
- Conditional Instructions: `addeq`, `ldreq`, etc in ARM