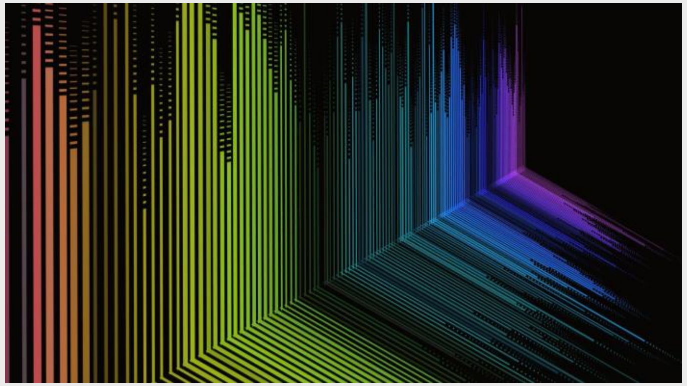
Conditional Branching

Lecture 8.3



Conditionals in programming languages

- Almost all programming languages have the capability to choose to do something only if a condition is met.
- Most commonly, this comes in the form of an “if” statement:

```
if (x < 2):
    x = 0
else:
    x = 3
```

But how does this translate to assembly code?

```
if (x < 2):
    x = 0
else:
    x = 3
```

But how does this translate to assembly code?

```
if (x < 2):
    x = 0
else:
    x = 3
```

First thing to notice: we have two mutually exclusive computations to perform!

But how does this translate to assembly code?

```
if (x < 2):
    x = 0
else:
    x = 3
```

Second thing to notice:
we have a condition that determines which instruction to *branch* to

But how does this translate to assembly code?

```
if (x < 2):
    x = 0
else:
    x = 3
```

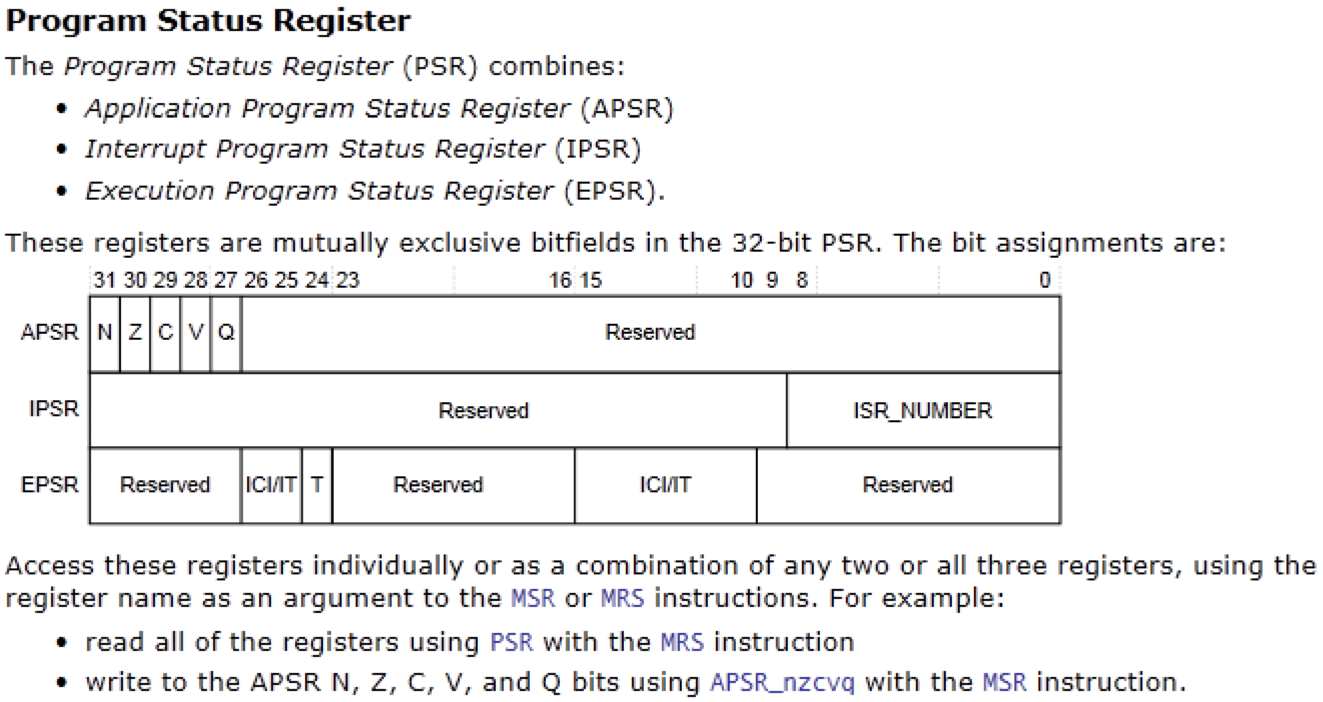
Second thing to notice:
 we have a condition that determines which instruction to *branch* to

This requires the ability to branch only when a specific condition is true

IF Tests: CMP

- In ARM assembly, the CMP (Compare) instruction allows values to be compared:
  - CMP R1, R2 subtracts the 2nd value (R2) from the 1st (R1)
  - The result of this subtraction is then used to update the **Application Program Status Register (APSR)**
    - Performed by the ALU
  - Specifically, 4 flag bits are updated within the APSR:
    - **N**	ALU result was **N**egative
    - **Z**	ALU result was **Z**ero
    - **C**	ALU set the **C**arry bit
    - **V**	ALU result caused o**V**erflow

APSR and the Program Status Register



Program Status Register

The *Program Status Register* (PSR) combines:

- *Application Program Status Register* (APSR)
- *Interrupt Program Status Register* (IPSR)
- *Execution Program Status Register* (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:

	31	30	29	28	27	26 25	24	23 16	15 10	9	8 0
APSR	N	Z	C	V	Q	Reserved					
IPSR	Reserved										ISR_NUMBER
EPSR	Reserved					ICI/IT	T	Reserved	ICI/IT	Reserved	

Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- read all of the registers using PSR with the MRS instruction
- write to the APSR N, Z, C, V, and Q bits using APSR_nzcvq with the MSR instruction.

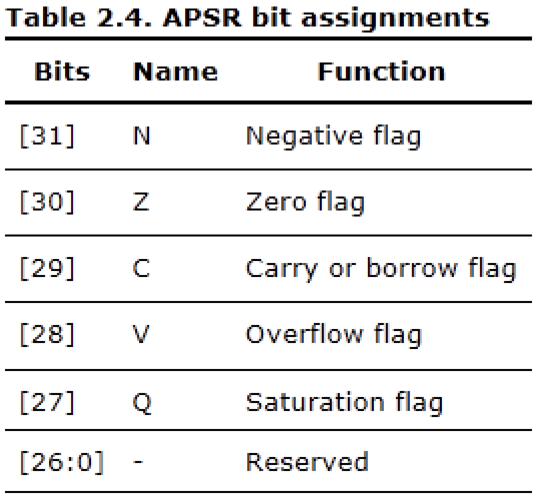
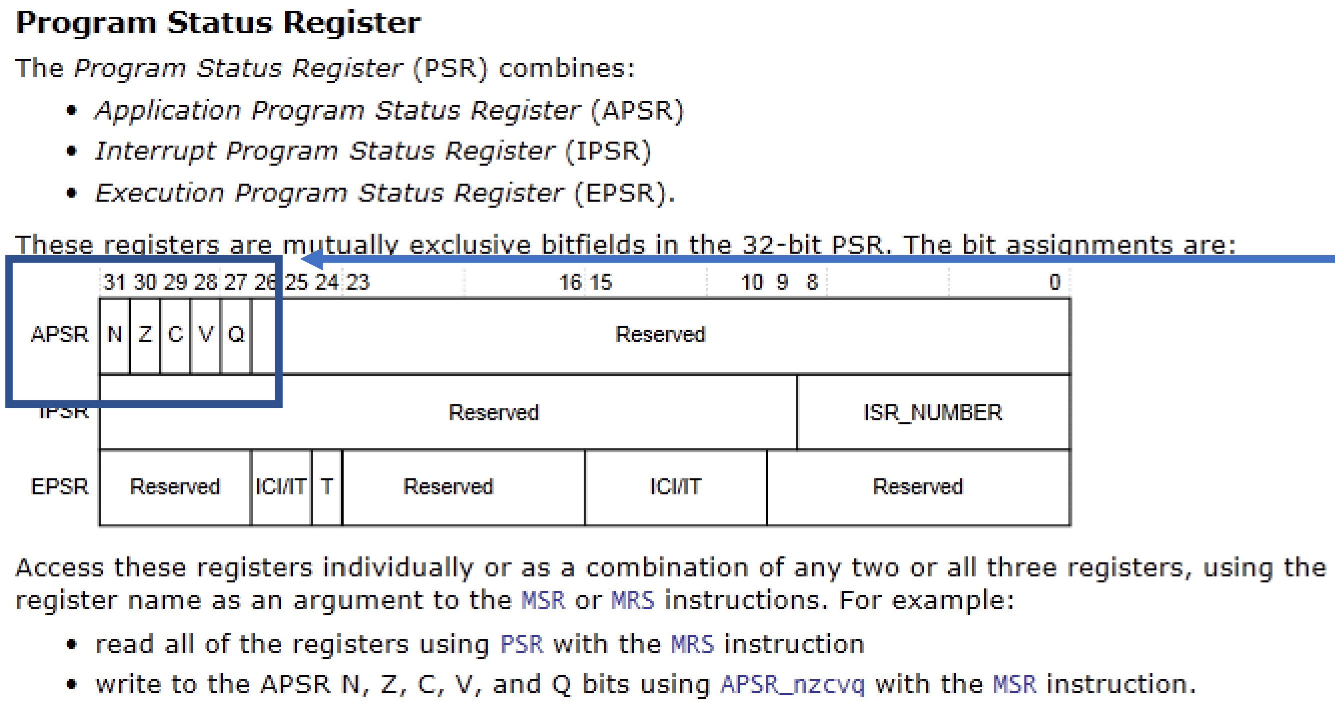


Table 2.4. APSR bit assignments

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	Saturation flag
[26:0]	-	Reserved

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/CHDBIBGJ.html

APSR and the Program Status Register



Program Status Register

The *Program Status Register* (PSR) combines:

- *Application Program Status Register* (APSR)
- *Interrupt Program Status Register* (IPSR)
- *Execution Program Status Register* (EPSR).

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are:

	31	30	29	28	27	26 25	24	23 16	15 10	9	8 0
APSR	N	Z	C	V	Q	Reserved					
IPSR	Reserved										ISR_NUMBER
EPSR	Reserved					ICI/IT	T	Reserved	ICI/IT	Reserved	

Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- read all of the registers using PSR with the MRS instruction
- write to the APSR N, Z, C, V, and Q bits using APSR_nzcvq with the MSR instruction.

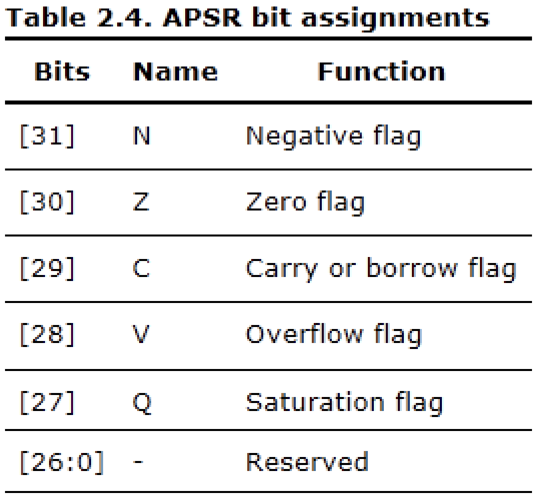
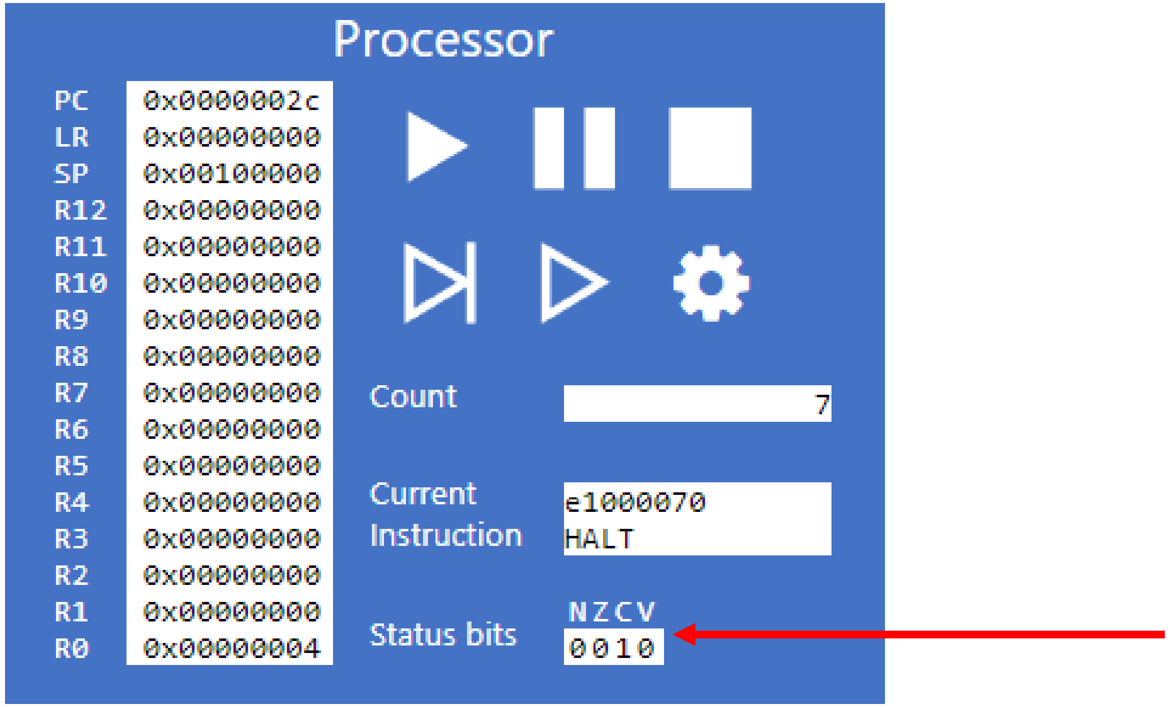


Table 2.4. APSR bit assignments

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	Saturation flag
[26:0]	-	Reserved

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/CHDBIBGJ.html

In ARMlite …



Processor

PC	0x0000002c
LR	0x00000000
SP	0x00100000
R12	0x00000000
R11	0x00000000
R10	0x00000000
R9	0x00000000
R8	0x00000000
R7	0x00000000
R6	0x00000000
R5	0x00000000
R4	0x00000000
R3	0x00000000
R2	0x00000000
R1	0x00000000
R0	0x00000004

Count 7

Current Instruction e1000070 HALT

Status bits NZCV 0010

You can visually inspect the outcome of a CMP here

details

- cmp r2,#1234

compare

register 2

compare r2 and #1234

r2 - #1234;
set APSR with
ALU flags

store the ALU flags in the APSR

Conditional Branching using the APSR

- Branch (B) reads the APSR and jumps according to the flags and the relevant suffix you give it.
- In ARMlite, there are five variants of **B**ranch:
    - **B** – unconditional branch
    - **BEQ** ‘Branch if EQual’
    - **BGT** ‘Branch if Greater Than’
    - **BLT** ‘Branch if Less Than’
    - **BNE** ‘Branch if Not Equal’
- In the ARM instruction set more generally, there are many others

Determining the comparison from the flags

In ARM assembly, the **condition code suffix** can be added to many operations. e.g. movne r1,#12

Suffix	Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not equal
CS or HS	C set	Higher or same (unsigned >=)
CC or LO	C clear	Lower (unsigned <)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned >)
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V differ	Signed <
GT	Z clear, N and V the same	Signed >
LE	Z set, N and V differ	Signed <=

SO .. how does this translate to assembly code?

```
if (x < 2):
    x = 0
```

SO .. how does this translate to assembly code?

```
if (x <= 2):
    x = 0
```

```
LDR R0,X    // assume X is holding a value
CMP R0,#2   // compare contents of R0 with #2
BGT skip    // if R0 > 2 then jump to skip
MOV R0,#0    // if R0 <= 2 then assign R0 #0
skip:        // continue program from here
…
```

What about this ?

```
if (x < 2):
    x = 0
else:
    x = 3
```

What about this ?

```
if (x <= 2):
    x = 0
else:
    x = 3
```

```
LDR R0,X        // assume X is holding a value
CMP R0,#2       // compare contents of R0 with #2
BGT else        // if R0 > 2 then jump to else
MOV R0,#0       // case for R0 <= 2
B cont          // Branch to label cont
else:
MOV R0,#3        // case for R0 > 2
cont:
 ….
```

What about this ?

```
if (x <= 2):
    x = 0
else:
    x = 3
```

```
LDR R0,X       // assume X is holding a value
CMP R0,#2      // compare contents of R0 with #2
BGT else       // if R0 > 2 then jump to label else
MOV R0,#0      // case for R0 <= 2
B cont         // Branch to label cont
else:
MOV R0,#3       // case for R0 > 2
cont:
 ….
```

Why do you think we need this branch ?

Pause the video and try this one

```
if (x <= 2):
    x = 0
else if (x <= 4):
    x = 2
else
    x = 4
```

Pause the video and try this one

```
if (x <= 2):
    x = 0
else if (x <= 4):
    x = 2
else
     x = 4
```

```
LDR R0,X              // assume X is holding a value
CMP R0,#2             // compare contents of R0 with #2
BGT else1             // if R0 > 2 then jump to label else
MOV R0,#0             // case for R0 <= 2
B cont                // Branch to label cont
else1:                // if R0 > 2
CMP R0,#4             // compare contents of R0 with #4
BGT else2             // if R0 > 4 then jump to label else2
MOV R0, #2            // otherwise, handled case for R0 > 2 <= 4
B cont                // Branch to label cont
else2:                // if R0 > 4
MOV R0,#4             // case for R0 > 4
cont:                 // exit point of conditionals. Continue with program from here
 ….
```

Pause the video and try this one

```
if (x <= 2):
     x = 0
else if (x <= 4):
     x = 2
else
     x = 4
```

```
LDR R0,X              // assume X is holding a value
CMP R0,#2             // compare contents of R0 with #2
BGT else1             // if R0 > 2 then jump to label else
MOV R0,#0             // case for R0 <= 2
B cont                // Branch to label cont
else1:                // if R0 > 2
CMP R0,#4             // compare contents of R0 with #4
BGT else2             // if R0 > 4 then jump to label else2
MOV R0, #2            // otherwise, handled case for R0 > 2 <= 4
B cont                // Branch to label cont
else2:                // if R0 > 4
MOV R0,#4             // case for R0 > 4
cont:                 // exit point of conditionals. Continue with program from here
 ….
```

So yeah - things get complicated quickly !