Control Flow Instructions
So Far...

- All instructions have allowed us to manipulate data
- So we’ve built a calculator
- In order to build a computer, we need ability to make decisions...
Labels

- Any instruction can be associated with a label
- Example:
  
  ```
  start    ADD r0,r1,r2 ; a = b+c
  next     SUB r1,r1,#1 ; b--
  ```

- In fact, every instruction has a label regardless if the programmer explicitly names it
  
  - The label is the address of the instruction
  - A label is a pointer to the instruction in memory
  - Therefore, the text label doesn’t exist in binary code
C Decisions: if Statements

- if statements in C
  - if (condition) clause
  - if (condition) clause1 else clause2

- Rearrange 2nd if into following:
  
  ```
  if (condition) goto L1;
  clause2;
  goto L2;
  L1: clause1;
  L2: 
  ```

- Not as elegant as if-else, but same meaning
The simplest control instruction is equivalent to a C goto statement

goto label (in C) is the same as:

B label (in ARM)

B is shorthand for “branch”. This is called an unconditional branch meaning that the branch is done regardless of any conditions.

There are also conditional branches
ARM also has variants of the branch instruction that only goto the label if a certain condition is TRUE

Examples:

- `BEQ label ; BRANCH EQUAL`
- `BNE label ; BRANCH NOT EQUAL`
- `BLE label ; BRANCH LESS THAN EQUAL`
- `BLT label ; BRANCH LESS THAN`
- `BGE label ; BRANCH GREATER THAN EQUAL`
- `BGT label ; BRANCH GREATER THAN`

Plus more …

The condition is T/F based upon the fields in the Program Status Register
Program Status Registers

- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation Overflowed

- **Sticky Overflow flag - Q flag**
  - Architecture 5TE/J only
  - Indicates if saturation has occurred

- **J bit**
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state

- **Interrupt Disable bits.**
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.

- **T Bit**
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state

- **Mode bits**
  - Specify the processor mode
Flags and Their Use

- The N flag
  - Set if the result is negative or equivalently if the MSB == ‘1’

- The Z flag
  - Set if the result is zero

- The C flag
  - Set if
    - The result of an addition is greater than $2^{32}$
    - The result of a subtraction is positive
    - Carryout from the shifter is ‘1’

- The V flag (oVerflow)
  - Set if there is overflow
The possible condition codes are listed below

- Note AL is the default and does not need to be specified

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
<th>Flags tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Equal</td>
<td>Z=1</td>
</tr>
<tr>
<td>NE</td>
<td>Not equal</td>
<td>Z=0</td>
</tr>
<tr>
<td>CS/HS</td>
<td>Unsigned higher or same</td>
<td>C=1</td>
</tr>
<tr>
<td>CC/LO</td>
<td>Unsigned lower</td>
<td>C=0</td>
</tr>
<tr>
<td>MI</td>
<td>Minus</td>
<td>N=1</td>
</tr>
<tr>
<td>PL</td>
<td>Positive or Zero</td>
<td>N=0</td>
</tr>
<tr>
<td>VS</td>
<td>Overflow</td>
<td>V=1</td>
</tr>
<tr>
<td>VC</td>
<td>No overflow</td>
<td>V=0</td>
</tr>
<tr>
<td>HI</td>
<td>Unsigned higher</td>
<td>C=1 &amp; Z=0</td>
</tr>
<tr>
<td>LS</td>
<td>Unsigned lower or same</td>
<td>C=0 or Z=1</td>
</tr>
<tr>
<td>GE</td>
<td>Greater or equal</td>
<td>N=V</td>
</tr>
<tr>
<td>LT</td>
<td>Less than</td>
<td>N!=V</td>
</tr>
<tr>
<td>GT</td>
<td>Greater than</td>
<td>Z=0 &amp; N=V</td>
</tr>
<tr>
<td>LE</td>
<td>Less than or equal</td>
<td>Z=1 or N=!V</td>
</tr>
<tr>
<td>AL</td>
<td>Always</td>
<td></td>
</tr>
</tbody>
</table>
The ARM Register Set

Only need to worry about cpsr (current program status register)

Current Visible Registers

Abort Mode

Banked out Registers

User

FIQ

IRQ

SVC

Undef
Compiling C if into ARM

- Compile by hand
  
  ```
  if (i == j) f=g+h;
  else f=g-h;
  ```

- Use this mapping:
  
  ```
  f: r0, g: r1, h: r2, i: r3, j: r4
  ```
Comparison Instructions

- In order to perform branch on the “==” operation we need a new instruction
- `CMP` – Compare: subtracts a register or an immediate value from a register value and updates condition codes
- Examples:
  - `CMP r3, #0 ; set Z flag if r3 == 0`
  - `CMP r3, r4 ; set Z flag if r3 == r4`

All flags are set as result of this operation, not just Z.
Compiling C if into ARM

Compile by hand

```c
if (i == j) f = g + h;
else f = g - h;
```

- Final compiled MIPS code:
  ```
  CMP r3, r4 ; Z = 1 if i==j
  BEQ True ; goto True when i==j
  SUB r0, r1, r2 ; f=g-h(false)
  B Fin ; goto Fin
  True ADD r0, r1, r2 ; f=g+h (true)
  Fin
  ```

- Note: Compiler automatically creates labels to handle decisions (branches) appropriately. Generally not found in C code.
Loops in C/Assembly

- Simple loop in C;
  ```c
  do {
    g--;  
    i = i + j;
  }  
  while (i != h);
  
  Rewrite this as:
  
  Loop: g--; 
  i = i + j;
  if (i != h) goto Loop;
  
  Use this mapping:
  g: r1, h: r2, i: r3, j: r4
Loops in C/Assembly

Final compiled MIPS code:

Loop
  SUB r1, r1, #1 ; g--
  ADD r3, r3, r4 ; i = i + j
  CMP r3, r2 ; cmp i, h
  BNE Loop ; goto Loop
            ; if i != h
Inequalities in ARM

- Until now, we’ve only tested equalities (== and != in C). General programs need to test < and > as well.
- Use CMP and BLE, BLT, BGE, BGT
- Examples:

  if (f < 10) goto Loop; => CMP r0,#10
  BLT Loop

  if (f >= i) goto Loop; => CMP r0,r3
  BGE Loop
There are three types of loops in C:
- while
- do... while
- for

Each can be rewritten as either of the other two, so the method used in the previous example can be applied to \texttt{while} and \texttt{for} loops as well.

Key Concept: Though there are multiple ways of writing a loop in ARM, conditional branch is key to decision making.
Choose among four alternatives depending on whether \( k \) has the value 0, 1, 2 or 3. Compile this C code:

```c
switch (k) {
    case 0: f=i+j; break; /* k=0*/
    case 1: f=g+h; break; /* k=1*/
    case 2: f=g-h; break; /* k=2*/
    case 3: f=i-j; break; /* k=3*/
}
```
Example: The C Switch Statement

- This is complicated, so simplify.
- Rewrite it as a chain of if-else statements, which we already know how to compile:

  ```c
  if (k==0) f = i + j;
  else if (k==1) f = g + h;
  else if (k==2) f = g - h;
  else if (k==3) f = i - j;
  ```

- Use this mapping:

  
  ```
  f: $s0, g: $s1, h: $s2, i: $s3, j: $s4, k: $s5
  ```
Example: The C Switch Statement

```c
CMP r5,#0 ; compare k, 0
BNE L1 ; branch k!=0
ADD r0,r3,r4 ; k==0 so f=i+j
B Exit ; end of case so Exit

L1 CMP r5,#1 ; compare k, -1
BNE L2
ADD r0,r1,r2 ; k==1 so f=g+h
B Exit ; end of case so Exit

L2 CMP r5,#2 ; compare k, 2
BNE L3 ; branch k!=2
SUB r0,r1,r2 ; k==2 so f=g-h
B Exit ; end of case so Exit

L3 CMP r5,#3 ; compare k, 3
BNE Exit ; branch k!=3
SUB r0,r3,r4 ; k==3 so f=i-j
```

Exit
Predicated Instructions

*All instructions can be executed conditionally. Simply add \{EQ, NE, LT, LE, GT, GE, etc.\} to end*

C source code

```c
if (r0 == 0) {
    r1 = r1 + 1;
} else {
    r2 = r2 + 1;
}
```

ARM instructions

**unconditional**

```asm
CMP r0, #0
BNE else
ADD r1, r1, #1
B end
else
    ADD r2, r2, #1
end
```

**conditional**

```asm
CMP r0, #0
ADDEQ r1, r1, #1
ADDNE r2, r2, #1
...```

- 5 instructions
- 5 words
- 5 or 6 cycles

- 3 instructions
- 3 words
- 3 cycles
Conclusions

- A Decision allows us to decide which pieces of code to execute at run-time rather than at compile-time.
- C Decisions are made using conditional statements within an if, while, do while or for.
- CMP instruction sets status register bits
- ARM Decision making instructions are the conditional branches: BNE, BEQ, BLE, BLT, BGE, BGT.
Conclusion

- Instructions so far:
  - Previously:
    - ADD, SUB, MUL, MOV,
    - LDR{H,B,SH,SB}, STR{H,B}
  - New:
    - CMP, B{EQ,NE,LT,LE,GT,GE}