ARM Registers

Register – internal CPU hardware device that stores binary data; can be accessed much more rapidly than a location in RAM

ARM has

13 general-purpose registers R0-R12
1 Stack Pointer (SP) – R13
1 Link Register (LR) – R14 holds the caller’s return address
1 Program Counter (PC) – R15
1 Current Program Status Register (CPSR)
Processor Status Register (PSR)

Contains: Condition flags that are set by arithmetic and logical CPU instructions and used for conditional execution

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>C</th>
<th>V</th>
<th>Q</th>
<th>GE</th>
</tr>
</thead>
</table>

- **N** (Negative/Less than)
- **Z** (Zero)
- **C** (Carry out)
- **V** (Overflow)
- **Q** (Sticky Overflow)
- **GE** (Greater than or equal to)

**Mode bits**

- Data Endianness bit

**Contains:**
- Condition flags that are set by arithmetic and logical CPU instructions and used for conditional execution.
Processor Status Register (PSR)

- The N, Z, C, and V bits are the condition code flags.
- Flags are set by arithmetic and logical CPU instructions and used for conditional execution.
- The processor tests these flags to determine whether to execute a conditional instruction.
  
  N – Negative / less than
  Z – Zero
  C – Carry out
  V – Overflow
ARM Instructions

- ARM instructions are written as an operation code (opcode), followed by zero or more operands.
- Operands may be constants, registers, or memory references.
ARM Instructions

• Simplified instruction syntax:
  opcode{cond}{flags} Rd, Rn, operand2
  where:
  – {cond} is an optional two-letter condition, e.g. EQ
  – {flags} is an optional additional flag, e.g. S
  – Rd is the destination register
  – Rn is the first source register
  – Operand2 is a flexible second operand

• Syntax is rigid (for the most part):
  – 1 operator, 3 operands
ARM Instructions

Note:

• Operand2 is a flexible second operand to most instructions
  – it is passed through the barrel shifter (a functional unit that can rotate and shift values)
  – it can take one of four forms:
    o Immediate value: an 8-bit number rotated right by an even number of places
    o Register
    o Register shifted by a value: a 5-bit unsigned integer shift.
    o Register shifted by a register: the bottom 8 bits of a register.
ARM Instructions

Examples of Operand2

- Immediate values
  - add r0, r1, #3
  - mov r0, #15
  - mov r1, #0x12

- Register shifted by a value
  - mov r0, r1, lsl #4
  - orr r1, r1, lsr #10

- Register shifted by a register
  - cmp r1, r2, lsl r0
  - add r5, r3, ror r0
One way to make your code more readable is to use comments!

The following symbols are acceptable:

- The // symbol - similar to how it is used in C
- The @, anything from @ to end of line is a comment and will be ignored
- The block comment, /* comment */
Labels in ARM

For our emulator, a label in ARM is designated as follows:
label_name:
e.g.
start:
...
loop:
...
done:
Immediates

- Immediates are numerical constants.

- They appear often in code, so there are ways to indicate their existence

- Add Immediate:

  ```
  /* f = g + 10      (in C)          */
  ADD r0, r1, #10   //  (in ARM)
  
  where ARM registers r0, r1 are associated with C variables f, g
  
  The second operand is a #number instead of a register.
  ```
Move instructions

ARM’s mov instruction is used to initialize a register.

- `mov Rd ← Operand2`
- `mvn Rd ← 0xFFFFFFFF EOR Operand2`

• Examples
  - `mov r0, #15`
  - `mov r0, r1`
Ldr pseudo-instruction: ldr

Sometimes the value to be placed into a register is too large for the mov instruction. For such cases, the ldr pseudo-instruction can be used.

• The ldr pseudo-instruction loads a register with either:
  – a 32-bit constant value
  – an address.

• General format:
  \text{LDR}\{\text{condition}\} \text{ register},=\left[ \text{expression} \mid \text{label-expression} \right]
**ldr pseudo-instruction**

- The *ldr* pseudo-instruction is used for two main purposes:
  - To generate literal constants when an immediate value cannot be moved into a register using a *mov* instruction because it is too large.
    
    e.g. `ldr r0, =44` @ puts the value 44 in register r0
    
    `ldr r0, =0xffff5555` @ puts the hex value in r0

  - To load a program-relative address or an external address into a register.
    
    e.g. `ldr r2, =fmt` @ puts the address of fmt in r2
Compare instructions

• cmp – compare
  o Flags set to result of (Rn – Operand2)

• cmn – compare negative
  o Flags set to result of (Rn + Operand2)

• tst – bitwise test
  o Rd := Rn or Operand2

• teq – test for equivalence
  o Rd := Rn and not Operand2
Compare instructions

• Comparisons produce no results – they just set condition codes.

• Ordinary instructions will also set condition codes if the “S” bit is set. The “S” bit is implied for comparison instructions.

• Examples of compare instructions
  
  cmp r0, r1
  cmp r0, #10
  tst r1, #1
  teq r0, 41
Logical instructions – Chapter 7 (pp 97 – 98)

• and – logical and
  \[ Rd ← Rn \text{ and } \text{Operand2} \]
• eor – exclusive or
  \[ Rd ← Rn \text{ eor } \text{Operand2} \]
• orr – logical or
  \[ Rd ← Rn \text{ or } \text{Operand2} \]
• bic – bitwise clear
  \[ Rd ← Rn \text{ and } \text{not } \text{Operand2} \]

• Examples
  and r2, r0, r1
    \[ // r2 = r0 \text{ and } r1 \]
  eor r2, r2, #1
    \[ // r2 = r0 \text{ ^ } 1 \]
Arithmetic instructions (p. 104)

- **add** \( R_d \leftarrow R_n + \text{Operand2} \)
- **adc** \( R_d \leftarrow R_n + \text{Operand} + \text{Carry} \)
- **sub** \( R_d \leftarrow R_n - \text{Operand2} \)
- **sbc** \( R_d \leftarrow R_n - \text{Operand2} - \text{not(Carry)} \)
- **rsb** \( R_d \leftarrow \text{Operand2} - R_n \)
- **rsc** \( R_d \leftarrow \text{Operand2} - R_n - \text{not(Carry)} \)

- **Examples**
  - `add r2, r0, r1`  //  \( r2 = r0 + r1 \)
  - `sub r2, r0, r1`  //  \( r2 = r0 - r1 \)
ARM instructions can be made to execute conditionally by postfixing them with the appropriate condition code field.

– This improves code density and performance by reducing the number of forward branch instructions.

```
cmp   r3, #0
beq   skip
add   r0, r1, r2
```

```
cmp   r3, #0
addne r0, r1, r2
```
By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using "S". CMP does not need "S".

```
loop:
    subs r1, r1, #1  # decrement r1 and set flags
    bne loop        # if Z flag clear then branch
```

"s" is necessary to set flags
Conditional execution examples

C source code

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

ARM instructions

**Unconditional**

- `CMP r0, #0`
- `BNE else`
- `ADD r1, r1, #1`
- `B done`

**Conditional**

- `CMP r0, #0`
- `ADDEQ r1, r1, #1`
- `ADDNE r2, r2, #1`

- `done:`

- 5 instructions
- 5 words
- 5 or 6 cycles

- 3 instructions
- 3 words
- 3 cycles