

ARM instruction set



- ARM versions.
- ARM assembly language.
- ARM programming model.
- ARM memory organization.
- ARM data operations.
- ARM flow of control.

ARM versions



- ARM architecture has been extended over several versions.
- We will concentrate on ARM7.

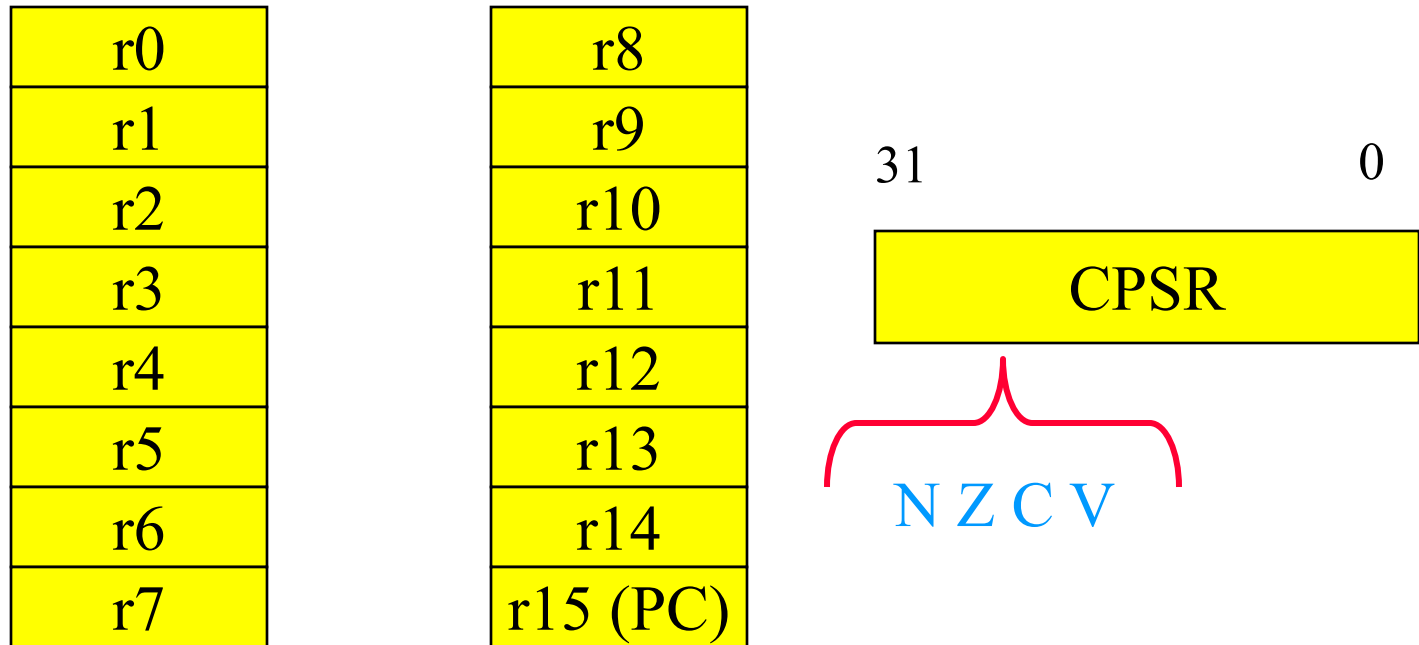
ARM assembly language



- Fairly standard assembly language:

```
LDR r0, [r8] ; a comment  
label ADD r4, r0, r1
```

ARM programming model



Endianness

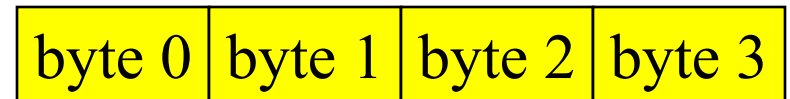
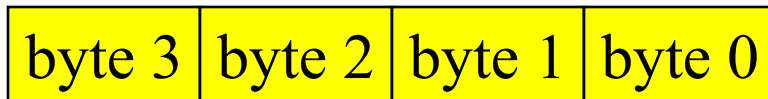
- Relationship between bit and byte/word ordering defines endianness:

bit 31

bit 0

bit 0

bit 31



little-endian

big-endian

ARM data types



- Word is 32 bits long.
- Word can be divided into four 8-bit bytes.
- ARM addresses can be 32 bits long.
- Address refers to byte.
 - Address 4 starts at byte 4.
- Can be configured at power-up as either little- or bit-endian mode.

ARM status bits

- Every arithmetic, logical, or shifting operation sets CPSR bits:
 - N (negative), Z (zero), C (carry), V (overflow).
- Examples:
 - $-1 + 1 = 0$: NZCV = 0110.
 - $2^{31}-1+1 = -2^{31}$: NZCV = 1001.

ARM data instructions



■ Basic format:

```
ADD r0, r1, r2
```

- Computes $r1+r2$, stores in $r0$.

■ Immediate operand:

```
ADD r0, r1, #2
```

- Computes $r1+2$, stores in $r0$.

ARM data instructions



- ADD, ADC : add (w. carry)
- SUB, SBC : subtract (w. carry)
- RSB, RSC : reverse subtract (w. carry)
- MUL, MLA : multiply (and accumulate)
- AND, ORR, EOR
- BIC : bit clear
- LSL, LSR : logical shift left/right
- ASL, ASR : arithmetic shift left/right
- ROR : rotate right
- RRX : rotate right extended with C

Data operation varieties



- Logical shift:
 - fills with zeroes.
- Arithmetic shift:
 - fills with ones.
- RRX performs 33-bit rotate, including C bit from CPSR above sign bit.

ARM comparison instructions



- CMP : compare
- CMN : negated compare
- TST : bit-wise AND
- TEQ : bit-wise XOR
- These instructions set only the NZCV bits of CPSR.

ARM move instructions



- MOV, MVN : move (negated)

```
MOV r0, r1 ; sets r0 to r1
```

ARM load/store instructions

- LDR, LDRH, LDRB : load (half-word, byte)
- STR, STRH, STRB : store (half-word, byte)
- Addressing modes:
 - register indirect : `LDR r0, [r1]`
 - with second register : `LDR r0, [r1, -r2]`
 - with constant : `LDR r0, [r1, #4]`

ARM ADR pseudo-op



- Cannot refer to an address directly in an instruction.
- Generate value by performing arithmetic on PC.
- ADR pseudo-op generates instruction required to calculate address:

```
ADR r1, FOO
```

Example: C assignments

■ C:

```
x = (a + b) - c;
```

■ Assembler:

```
ADR r4,a           ; get address for a
LDR r0,[r4]        ; get value of a
ADR r4,b           ; get address for b, reusing r4
LDR r1,[r4]        ; get value of b
ADD r3,r0,r1       ; compute a+b
ADR r4,c           ; get address for c
LDR r2,[r4]        ; get value of c
```

C assignment, cont'd.



```
SUB r3,r3,r2      ; complete computation of x
ADR r4,x          ; get address for x
STR r3,[r4]       ; store value of x
```


Example: C assignment

■ C:

```
y = a * (b + c);
```

■ Assembler:

```
ADR r4,b ; get address for b
LDR r0,[r4] ; get value of b
ADR r4,c ; get address for c
LDR r1,[r4] ; get value of c
ADD r2,r0,r1 ; compute partial result
ADR r4,a ; get address for a
LDR r0,[r4] ; get value of a
```

C assignment, cont'd.



```
MUL r2,r2,r0 ; compute final value for y  
ADR r4,y ; get address for y  
STR r2,[r4] ; store y
```

Example: C assignment

■ C:

```
z = (a << 2) | (b & 15);
```

■ Assembler:

```
ADR r4,a ; get address for a  
LDR r0,[r4] ; get value of a  
MOV r0,r0,LSL 2 ; perform shift  
ADR r4,b ; get address for b  
LDR r1,[r4] ; get value of b  
AND r1,r1,#15 ; perform AND  
ORR r1,r0,r1 ; perform OR
```

C assignment, cont'd.



ADR r4,z ; get address for z

STR r1,[r4] ; store value for z

Additional addressing modes

- Base-plus-offset addressing:

```
LDR r0, [r1, #16]
```

- Loads from location $r1+16$

- Auto-indexing increments base register:

```
LDR r0, [r1, #16]!
```

- Post-indexing fetches, then does offset:

```
LDR r0, [r1], #16
```

- Loads $r0$ from $r1$, then adds 16 to $r1$.

ARM flow of control



- All operations can be performed conditionally, testing CPSR:
 - EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE
- Branch operation:
 - B #100
 - Can be performed conditionally.

Example: if statement

■ C:

```
if (a > b) { x = 5; y = c + d; } else x = c - d;
```

■ Assembler:

```
; compute and test condition
```

```
ADR r4,a ; get address for a
```

```
LDR r0,[r4] ; get value of a
```

```
ADR r4,b ; get address for b
```

```
LDR r1,[r4] ; get value for b
```

```
CMP r0,r1 ; compare a < b
```

```
BLE fblock ; if a >= b, branch to false block
```

If statement, cont'd.



```
; true block
MOV r0,#5 ; generate value for x
ADR r4,x ; get address for x
STR r0,[r4] ; store x
ADR r4,c ; get address for c
LDR r0,[r4] ; get value of c
ADR r4,d ; get address for d
LDR r1,[r4] ; get value of d
ADD r0,r0,r1 ; compute y
ADR r4,y ; get address for y
STR r0,[r4] ; store y
B after ; branch around false block
```


If statement, cont'd.



```
; false block
fblock ADR r4,c ; get address for c
      LDR r0,[r4] ; get value of c
      ADR r4,d ; get address for d
      LDR r1,[r4] ; get value for d
      SUB r0,r0,r1 ; compute a-b
      ADR r4,x ; get address for x
      STR r0,[r4] ; store value of x
after ...
```

Example: Conditional instruction implementation

```
; true block
MOVLT r0,#5 ; generate value for x
ADRLT r4,x ; get address for x
STRLT r0,[r4] ; store x
ADRLT r4,c ; get address for c
LDRLT r0,[r4] ; get value of c
ADRLT r4,d ; get address for d
LDRLT r1,[r4] ; get value of d
ADDLT r0,r0,r1 ; compute y
ADRLT r4,y ; get address for y
STRLT r0,[r4] ; store y
```

Conditional instruction implementation, cont'd.



```
; false block
ADRGE r4,c ; get address for c
LDRGE r0,[r4] ; get value of c
ADRGE r4,d ; get address for d
LDRGE r1,[r4] ; get value for d
SUBGE r0,r0,r1 ; compute a-b
ADRGE r4,x ; get address for x
STRGE r0,[r4] ; store value of x
```

Example: switch statement

■ C:

```
switch (test) { case 0: ... break; case 1: ... }
```

■ Assembler:

```
ADR r2,test ; get address for test
```

```
LDR r0,[r2] ; load value for test
```

```
ADR r1,switchtab ; load address for switch table
```

```
LDR r1,[r1,r0,LSL #2] ; index switch table
```

```
switchtab DCD case0
```

```
DCD case1
```

...

Example: FIR filter



■ C:

```
for (i=0, f=0; i<N; i++)  
    f = f + c[i]*x[i];
```

■ Assembler

```
; loop initiation code  
MOV r0,#0 ; use r0 for I  
MOV r8,#0 ; use separate index for arrays  
ADR r2,N ; get address for N  
LDR r1,[r2] ; get value of N  
MOV r2,#0 ; use r2 for f
```

FIR filter, cont'.d

```
ADR r3,c ; load r3 with base of c
ADR r5,x ; load r5 with base of x
; loop body
loop LDR r4,[r3,r8] ; get c[i]
LDR r6,[r5,r8] ; get x[i]
MUL r4,r4,r6 ; compute c[i]*x[i]
ADD r2,r2,r4 ; add into running sum
ADD r8,r8,#4 ; add one word offset to array index
ADD r0,r0,#1 ; add 1 to i
CMP r0,r1 ; exit?
BLT loop ; if i < N, continue
```

ARM subroutine linkage



- Branch and link instruction:

```
BL foo
```

- Copies current PC to r14.

- To return from subroutine:

```
MOV r15, r14
```

Nested subroutine calls

■ Nesting/recursion requires coding convention:

```
f1    LDR r0,[r13] ; load arg into r0 from stack
      ; call f2()
      STR r14,[r13]! ; store f1's return adrs
      STR r0,[r13]! ; store arg to f2 on stack
      BL f2 ; branch and link to f2
      ; return from f1()
      SUB r13,#4 ; pop f2's arg off stack
      LDR r13!,r15 ; restore register and return
```


Summary



- Load/store architecture
- Most instructions are RISCy, operate in single cycle.
 - Some multi-register operations take longer.
- All instructions can be executed conditionally.