ARM Assembly Language Examples & Assembler

ARM Assembly Language Examples

Example 1: C to ARM Assembler

- C:

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

- ARM:

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


Example 2: C to ARM Assembler

- C:

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

- ARM:

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

Example 3: C to ARM Assembler

- C:

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

- ARM:

```
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
ADR r4,z           ; get address for z
STR r1,[r4]        ; store value for z
```


Example 4: Condition Codes

C:

```
if (i == 0)
 {
    i = i +10;
 }
```

ARM: (assume i in R1)

```
SUBS    R1, R1, #0
ADDEQ   R1, R1, #10
```


Example 5: Condition Codes

C:

```
for ( i = 0 ; i < 15 ; i++)
  {
     j = j + j;
  }
```

ARM:

```
       SUB   R0, R0, R0   ; i -> R0 and i = 0
start  CMP   R0, #15      ;   is i < 15?
       ADDLT R1, R1, R1   ;   j = j + j
       ADDLT R0, R0, #1   ;   i++
       BLT   start
```


Example 6: if statement [1]

- C:

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

- ARM:

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

Example 6: if statement [2]

```
; 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
```


Example 6: if statement [3]

```
; 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 6: Heavy Conditional Instruction Use [1]

Same C code; different ARM implementation

ARM:

```
; Compute and test the condition
ADR r4,a        ; get address for a
LDR r0,[r4]     ; get value of a
ADR r4,b        ; get address
```


Example 6: Heavy Conditional Instruction Use [2]

```
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
; false block
ADRGE r4,c        ; get address for c
```

Example 6: Heavy Conditional Instruction Use [3]

```
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
```


ARM Assembler

Assembly Language Basics

- The following is a simple example which illustrates some of the core constituents of an ARM assembler module:

```
        AREA Example, CODE, READONLY      ; name this block of code
        ENTRY                             ; mark first instruction
                                          ; to execute
start
        MOV     r0, #15                   ; Set up parameters
        MOV     r1, #20
        BL      firstfunc                 ; Call subroutine
        SWI     0x11                      ; terminate
firstfunc                                 ; Subroutine firstfunc
        ADD     r0, r0, r1                ; r0 = r0 + r1
        MOV     pc, lr                    ; Return from subroutine
                                          ; with result in r0
        END                               ; mark end of file
```

label, opcode, operands, comment

General Layout

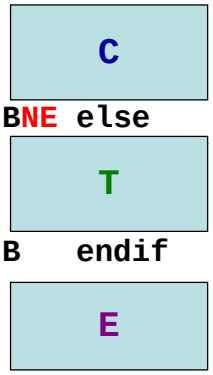
- The general form of lines in an assembler module is:

  label <space> opcode <space> operands <space> ; comment

- Each field must be separated by one or more <whitespace> (such as a space or a tab).
- Actual instructions never start in the first column, since they must be preceded by whitespace, even if there is no label.
- All three sections are optional and the assembler will also accept blank lines to improve the clarity of the code.

If statements

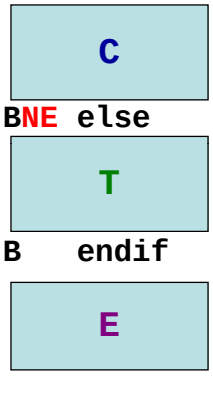
```
if C then T else E
```




```
// find maximum
if (R0>R1) then R2:=R0
else R2:=R1
```

If statements

```
if C then T else E
```




```
// find maximum
if (R0>R1) then R2:=R0
else R2:=R1

        CMP R0, R1
        BLE else
        MOV R2, R0
        B   endif
else:   MOV R2, R1
endif:
```

If statements

Two other options:

```
CMP   R0, R1
MOVGT R2, R0
MOVLE R2, R1

MOV   R2, R0
CMP   R0, R1
MOVLE R2, R1
```

```
// find maximum
if (R0>R1) then R2:=R0
else R2:=R1

        CMP R0, R1
        BLE else
        MOV R2, R0
        B   endif
else:   MOV R2, R1
endif:
```

If statements

```
if (R1==1 || R1==5 || R1==12) R0=1;

TEQ   R1, #1        ...
TEQNE R1, #5        ...
TEQNE R1, #12       ...
MOVEQ R0, #1        BNE fail
```

If statements

```
if (R1==0) zero
else if (R1>0) plus
else if (R1<0) neg

      TEQ   R1, #0
      BMI   neg
      BEQ   zero
      BPL   plus
neg:  ...
      B exit
Zero: ...
      B exit
      ...
```

If statements

```
R0=abs(R0)

TEQ    R0, #0
RSBMI  R0, R0, #0
```

Multi-way branches

```
        CMP R0, #`0'
        BCC other   @ less than '0'
        CMP R0, #`9'
        BLS digit   @ between '0' and '9'
        CMP R0, #`A'
        BCC other
        CMP R0, #`Z'
        BLS letter  @ between 'A' and 'Z'
        CMP R0, #`a'
        BCC other
        CMP R0, #`z'
        BHI other  @ not between 'a' and 'z'
letter: ...
```

Switch statements

```
switch (exp) {
  case c1: S1; break;
  case c2: S2; break;
  ...
  case cN: SN; break;
  default: SD;
}
```

```
e=exp;
if (e==c1) {S1}
else
  if (e==c2) {S2}
  else
  ...
```

Switch statements

```
switch (R0) {
  case 0: S0; break;
  case 1: S1; break;
  case 2: S2; break;
  case 3: S3; break;
  default: err;
}
```

The range is between 0 and N

Slow if N is large

```
        CMP R0, #0
        BEQ S0
        CMP R0, #1
        BEQ S1
        CMP R0, #2
        BEQ S2
        CMP R0, #3
        BEQ S3
err:    ...
        B exit
S0:     ...
        B exit
```

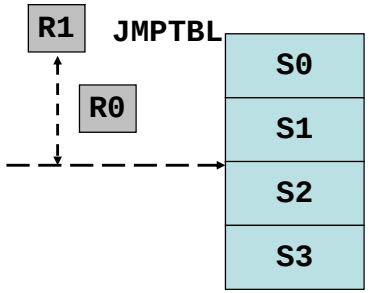
Switch statements

```
    ADR   R1, JMPTBL
    CMP   R0, #3
    LDRLS PC, [R1, R0, LSL #2]
err:...
    B     exit
S0: ...

JMPTBL:
    .word S0
    .word S1
    .word S2
    .word S3
```

What if the range is between M and N?

For larger N and sparse values, we could use a hash function.



Iteration

- repeat-until
- do-while
- for

repeat loops

```
do {S} while ( C )
```

```
loop:
        S
        C
        BEQ   loop
endw:
```