Computer Architecture Exercises

ARM Assembly Language

Adrian Wälchli

April 17, 2018

Contents

Registers

General purpose

`R0, R1, ..., R12`

Argument values

`R0, R1, R2, R3`

Return values

`R0, R1, R2, R3`

Special

Stack pointer `SP`, link register `LR`, program counter `PC`

Load

With register

Load data stored at address `R1` into `R0`

```
LDR R0, R1
```

With offset

Load address = `R1` + offset

Offset can also be a register

```
LDR R0, [R1, #4]
```

With pointer

X is a pointer (label) to some data

```
LDR R0, X
```

Store

With register

Store data in `R0` to address in `R1`

```
STR R0, R1
```

With offset

Store address = `R1` + offset

Offset can also be a register

```
STR R0, [R1, #8]
```

Move

Copy

Move data in register `R1` to `R0`

```
MOV R0, R1
```

Immediate

Move immediate value to register `R0`

```
MOV R0, #42
```

Arithmetic Operations

Common operations

Apply operation to R1 and R2 and save result in R0
R2 can also be immediate value

```
ADD  R0, R1, R2
SUB  R0, R1, #4
MUL  R0, R1, R2
SDIV R0, R1, #-3
UDIV R0, R1, R2
```

Division can be signed or unsigned

Shift

Logical shift left

Shift value in `R1` by `R2` to the left and save result in `R0`
`R2` can be immediate value

```
LSL R0, R1, R2
LSL R0, R1, #4
```

Logical shift right

```
LSR R0, R1, R2
LSR R0, R1, #1
```

Logical Operations

Bitwise AND

Apply bitwise and-operation to R1 and R2 and save result in R0

```
AND R0, R1, R2
AND R0, R1, #0xFF
```

Bitwise OR

Similar to AND

```
ORR R0, R1, R2
ORR R0, R1, #0b00010001
```

Exclusive OR (XOR)

```
EOR R0, R1, R2
```

Branch

Regular branch

PC = label

```
B label
```

Branch and link

PC = label, LR = address of next instruction

```
BL label
```

Condition Flags

There are four condition flags (binary) managed by ALU

- **N** Negative: Result of previous operation was negative
- **Z** Zero: Result of previous operation was zero
- **C** Carry (unsigned overflow): Result overflows 32-bit register
- **V** (Signed) overflow: Result overflows 32-bit signed number

Most instruction can be forced to update flags by appending "S" to their name (when applicable).

Compare

Update condition flags on `R0 - R1`

```
CMP R0, R1
CMP R0, #0
```

Condition Fields

All instructions can have *condition codes* appended to their names. An instruction is only executed when the condition is satisfied.

EQ	Equal
NE	Not equal
LT	Less than (signed)
GT	Greater than (signed)
LE	Less than or equal (signed)
GE	Greater than or equal (signed)

Condition Fields

Example 1

```
MOV     R0, #3
MOV     R1, #2
CMP     R0, R1
ADDEQ   R2, R0, #100
```

Condition is not satisfied, since $3 \neq 2$

Example 2

```
MOV     R0, #31
MULS    R0, R0, #-1
BLLT    mylabel
```

Condition is satisfied ($-1 \cdot 31 < 0$): branch is taken

Flexible Operand

Many instructions allow the last operand to be "flexible", e.g.

```
MOV R0, <Operand2>
```

where <Operand2> can be

- `R1, #<imm>`
- `R1, LSL R2`
- `R1, LSL #<imm>`
- `R1, LSR R2`
- `R1, LSR #<imm>`

Applies to arithmetic- and logical operations, and more.

Flexible Operand

Example

Shift contents of `R1` left by 1 bit and move it to `R0`
`R1` is unchanged

```
MOV R0, R1, LSR #1
```

Subroutines

Basic

```
// some code here

BL myfunc

// other code follows

myfunc:
// do something here
// ...
MOV PC, LR          // return
```

Subroutines

Arguments and return values

```
MOV R0, #1
MOV R1, #2
BL myfunc

// other code follows

myfunc:
ADD R9, R0, R1   // R0 and R1 are arguments
// do something else with R9
MOV R0, R9       // return value saved in R0
MOV PC, LR       // return
```

Problem: After returning, original value of R9 is lost. Also, can not recursively call subroutines.

Subroutines

Saving registers

```
myfunc:
// push register contents onto stack
STMDB SP!, {R9, R10, LR}

// do something with register R9 and R10
ADD R9, R0, R1
// do something else with R10

// save return value in R0
MOV R0, R9

// restore registers and return
LDMIA SP!, {R9, R10, PC}
```

Resources

- ARM Quick Reference Sheet:
  `http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001l/QRC0001_UAL.pdf`