Addressing Modes (3A)

Young Won Lim
1/27/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Young Won Lim
1/27/20

Based on

D.A. Patterson & J.H. Hennessy, Computer Organization and Design
(ARM ed)

Addressing Modes

1. Immediate	**ADD r2, r0, #5**
2. Register	**ADD r2, r0, r1**
3. Scaled register	**ADD r2, r0, r1, LSL #2**
4. Immediate offset pre-indexed without writeback	**LDR r2, [r0, #4]**
5. Register offset pre-indexed without writeback	**LDR r2, [r0, r1]**
6. Scaled register offset pre-indexed without writeback	**LDR, r2, [r0, r1, LSL #2]**
7. Immediate offset pre-indexed <u>with</u> writeback	**LDR r2, [r0, #4]!**
8. Reigster offset pre-indexed <u>with</u> writeback	**LDR r2, [r0, r1]!**
9. Scaled register offset pre-indexed <u>with</u> writeback	**LDR r2, [r0, r1, LSL #2]!**
10. Immediage offset post-indexed	**LDR r2, [r0] #4**
11. Register offset post-indexed	**LDR r2, [r0], r1**
12. Register offset post-indexed	**LDR r2, [r0], r1, LSL #2**

Addressing mode examples (1)

1. Immediate

ADD r2, r0, #5 r2 ← r0 + 5

2. Register

ADD r2, r0, r1 r2 ← r0 + r1

3. Scaled register

ADD r2, r0, r1, LSL #2 r2 ← r0 + (r1 << 2)

1. Immediate Operand

ADD r2, r0, #5 **r2 ← r0 + 5**

1st operand : register r0
2nd operand : immediate value #5

2. Register Operand

ADD r2, r0, r1 **r2 ← r0 + r1**

1st operand : register r0
2nd operand : register r1

3. Scaled Register Operand

ADD r2, r0, r1, LSL #2 **r2 ← r0 + (r1 << 2)**

1st operand : register r0
2nd operand : register r1 << 2

Addressing mode examples (2)

4. Immediate offset pre-indexed <u>without</u> writeback

LDR r2, [r0, #4]

adr ← **r0 + 4;**
r2 ← M[*adr*]

5. Register offset pre-indexed <u>without</u> writeback

LDR r2, [r0, r1]

adr ← **r0 + r1;**
r2 ← M[*adr*]

6. Scaled register offset pre-indexed <u>without</u> writeback

LDR, r2, [r0, r1, LSL #2]

adr ← **r0 + (r1<<2);**
r2 ← M[*adr*]

4. Immediate Offset Pre-index without Writeback

LDR r2, [r0, #4] *adr* ← **r0 + 4;** **r2 ← M[*adr*]**

adding operation before a memory access
base register r0 + immediate offset #4

the added address is used for a memory access

without the **!** suffix, the base register r0 is not updated

when traversing an array sequentially

PC relative addressing

5. Register Offset Pre-indexed without Writeback

LDR r2, [r0, r1] *adr* ← **r0 + r1;** **r2 ← M[*adr*]**

adding operation before a memory access
base register r0 + offset register r1

the added address is used for a memory access

without the **!** suffix, the base register r0 is not updated

index into an array
- array – base
- index – offset

6. Scaled Register Offset Pre-indexed without Writeback

LDR, r2, [r0, r1, LSL #2] *adr* ← **r0 + (r1 << 2); r2 ← M[***adr***]**

adding operation before a memory access
base register r0 + (offset register r1 << 2)

the added address is used for a memory access

without the **!** suffix, the base register r0 is not updated

the offset register r1 is never changed

index into an array
making an array index into a byte address
- array – base
- index – offset
- index * 4 – byte offset address

Addressing mode examples (3)

7. Immediate offset pre-indexed <u>with</u> writeback

LDR r2, [r0, #4]! r0 ← r0 + 4;
r2 ← M[r0]

8. Reigster offset pre-indexed <u>with</u> writeback

LDR r2, [r0, r1]! r0 ← r0 + r1;
r2 ← M[r0]

9. Scaled register offset pre-indexed <u>with</u> writeback

LDR r2, [r0, r1, LSL #2]! r0 ← r0 + (r1 << 2);
r2 ← M[r0]

7. Immediate Offset Pre-index with Writeback

LDR r2, [r0, #4]! **r0 ← r0 + 4;** **r2 ← M[r0]**

adding operation before a memory access
base register r0 + immediate offset #4

the added address is used for a memory access

base register r0 is updated with the added address

8. Register Offset Pre-indexed with Writeback

LDR r2, [r0, r1]! **r0 ← r0 + r1;** **r2 ← M[r0]**

adding operation before a memory access
base register r0 + offset register r1

the added address is used for a memory access

base register r0 is updated with the added address

9. Scaled Register Offset Pre-indexed with Writeback

LDR, r2, [r0, r1, LSL #2]! **r0 ← r0 + (r1 << 2); r2 ← M[r0]**

adding operation before a memory access
base register r0 + (offset register r1 << 2)

the added address is used for a memory access

base register r0 is updated with the added address

the offset register r1 is never changed

Addressing mode examples (4)

10. Immediage offset post-indexed

LDR r2, [r0] #4	**r2 ← M[r0];**
	r0 ← r0 + 4

11. Register offset post-indexed

LDR r2, [r0], r1	**r2 ← M[r0];**
	r0 ← r0 + r1

12. Register offset post-indexed

LDR r2, [r0], r1, LSL #2	**r2 ← M[r0];**
	r0 ← r0 + (r1 << 2)

10. Immediate Offset Post-index

LDR r2, [r0], #4 **r2 ← M[r0]; r0 ← r0 + 4**

first accessing memory, then adding operation
base register r0 + immediate offset #4

the initial base register r0 is used for a memory access

<u>no need</u> the **!** suffix, the base register is <u>always</u> <u>updated</u>

have similar applications like pre-index

11. Register Offset Post-index

LDR r2, [r0], r1 **r2 ← M[r0]; r0 ← r0 + r1**

first accessing memory, then adding operation
base register r0 + offset register r1

the initial base register r0 is used for a memory access

<u>no need</u> the **!** suffix, the base register r0 is <u>always</u> <u>updated</u>

have similar applications like pre-index

12. Scaled Register Offset Post-index

LDR r2, [r0], r1, LSL #2 **r2 ← M[r0]; r0 ← r0 + (r1 << 2)**

first accessing memory, then adding operation
base register r0 + (offset register r1 << 2)

the initial base register r0 is used for a memory access

<u>no need</u> the **!** suffix, the base register r0 is <u>always</u> <u>updated</u>

the offset register r1 is never changed

have similar applications like pre-index

Register Indirect Addressing

register indirect addressing :

the **location** of an **operand** is held in a **register**.
also called **indexed addressing** or **base addressing**.

```
registers
r0 = ABCD EFAB
r1 = 0123 4560
```

```
address              data
0123 4560     :   ABCD EFAB
```

```
M[0123 4560] = ABCD EFAB
       [r1]  = r0
        equivalence
```

r1 holds the address of a memory location
r0 holds the data at that location

www.cs.uregina.ca › pub › class › ARM-addressing › lecture
http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

Register Indirect Addressing – LDR

To **load** a value from memory into a register
using register-indirect addressing,
the **base register** is used

This **base register** holds the actual memory **address**

The **LDR** instruction inspects the **base register**,
interprets its value as the memory **address**,
fetches the **value** stored at that **address location**,
and then loads it into a **destination register**.

```
LDR    r0, ⟸ [r1]
; r0 receives the value held at the memory address pointed to by r1
; r0 is the destination register, r1 is the base register
```

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

Register Indirect Addressing – STR

To **store** a value to memory from a register
using register-indirect addressing,
a **base register** is again employed
to hold the actual memory address.

The **STR** instruction inspects the **base register**,
interprets its value as a memory **address location**,
and places the **value** held in the **source register**
into the **memory location**.

```
STR    r0, ➡ [r1]
; the memory location pointed to by r1 receives the value held in r0
; r0 is the source register, r1 is the base register
```

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

Loading a full 32-bit address

register-indirect addressing is simple
but has the following problem

How can a 32-bit address be loaded
into a register in the first place?

it might seem that a **MOV** instruction
would resolve this issue.

but all ARM instructions are 32 bits long,
bits are needed for the opcode and the destination register
less than 32 bits are left for an address

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

Pseudo-instruction ADR

The ARM uses a **pseudo-instruction**,
that does not have its own binary encoded instruction.

Instead the **assembler** translates this **pseudo-instruction**
into one or more real **instructions**.

ADR is one of the **pseudo-instruction**,
which loads an **address** into a **destination register**.

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

PC-Relative Addressing Example

```
Copycode ADR    r1, SRC    ; the value of r1 points to the SRC location
         ADR    r2, DST    ; the value of r2 points to the DST location
         LDR    r0, [r1]   ; load value at r1 address into r0 (SRC)
         STR    r0, [r2]   ; store value in r0 into r2 address (DST)
      .
SRC      .    ; source of data
         .
DST      .    ; destination for the data
```

`ADR r1, SRC`	will be converted into	`ADD r1, pc, #offset_src`
`ADR r2, DST`	will be converted into	`ADD r2, pc, #offset_dst`

PC-relative offset = Desired Address – (Current ADR Inst Address + 8)

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

PC-relative offset

```
0000 4000:      ADR r1, SRC      ; ADD r1, pc, #0x78
0000 4004:      ADR r2, DST      ; ADD r2, pc, #0x80


0000 4080:      XXXX XXXX

0000 408c:      YYYY YYYY
```

PC-relative offset = desired address – (Current ADR Inst Address + 8)
= desired address – (Current value of PC)

78 = 4080 – (4000+8) **(4000+8) + 78 = 4080**
80 = 408c – (4004+8) **(4004+8) + 80 = 408c**

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

Current value of PC

The trick with the **ADR** instruction relies on the fact that
the current value of the **PC** (**r15**) (8-byte advance)
will normally be *close* to the intended memory address location.

Thus an **ADR** instruction is translated
into one or more instructions
that can add a constant value to or
subtract a constant value
from the current value of the **PC**
and place the result in the destination register
specified by the original ADR instruction.

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

PC-Relative Addressing and Offset

PC-relative addressing :

The constant value is known as the **PC-relative offset**.
It can be calculated by the formula below:

PC-relative offset = desired address – (ADR Inst Address + 8)

The +8 in the formula is a consequence of
how the ARM processes instructions
using "pipeline" techniques

http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture4/lecture4-2-3.html

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf