COS10004 Computer Systems
Assignment 2

Daniel Coady (102084174) – 12:30 Wednesday

14/10/2019

1 mov

1.1 Syntax

```
mov x, y
```

Where:

- x is the destination
- y is the value

1.2 Description

Used to move a value into a register. Note that values must have 24 consecutive zeroes in it's binary notation.

1.3 Example

```
mov r0, $3F0000 ; valid
mov r0, $003F00 ; valid
mov r0, $00003F ; valid
mov r0, $300F00 ; invalid
```

2 orr

2.1 Syntax

```
orr x, y
```

Where:

- x is value 1 and the destination
- y is value 2

2.2 Description

Performs a bitwise OR operation on x and y, storing the result in x.

2.3 Example

```
mov r0, $10 ; r0 has 0x10
orr r0, $01 ; r0 has 0x11
```

3 eor

3.1 Syntax

```
eor x, y, z
```

Where:

- x is the destination register
- y is register holding the first value
- z is the second value

3.2 Description

Performs a bitwise exclusive OR operation on y and z, storing the result in x if specified. If x is not specified then the result is stored in y.

3.3 Example

```
eor r0, r1, #7
eor r0, #7
```

4 orn

4.1 Syntax

```
orn x, y, z
```

Where:

- x is the destination register
- y is register holding the first value
- z is the second value

4.2 Description

Performs a bitwise OR NOT operation on y and z, storing the result in x if specified. If x is not specified then the result is stored in y.

4.3 Example

```
orn r0, r1, #7
orn r0, #7
```

5 and

5.1 Syntax

```
and x, y, z
```

Where:

- x is the destination
- y is the register holding the first value
- z is the second value

5.2 Description

Performans a bitwise AND operation on y and z. Stores the result in x, but if x is not specified then it is stores in y.

5.3 Example

```
and r0, r1, #7
and r0, #7
```

6 ldr

6.1 Syntax

```
ldr x, [y]
```

Where:

- x is the register to store the value in
- y is the location to get the value from

6.2 Description

A pseudo instruction for storing 32-bit values in memory.

6.3 Example

7 ldrd

7.1 Syntax

```
ldrd x, y, [z]
```

Where:

- x is the register for the least significant half of the value
- y is the register for the most significant half of the value
- z is the location to get the value from

7.2 Description

Allows for storing of a 64-bit value across 2 32-bit registers.

7.3 Example

```
ldrd r0, r1, [r2, #4]
```

8 str

8.1 Syntax

```
str x, [y]
```

Where:

- x is the value to store
- y is the location to store the value into

8.2 Description

Used to store values within registers.

8.3 Example

```
str r0, [r1, #4]
```

9 add

9.1 Syntax

```
add x, y, z
add y, z
```

Where:

- x is the destination for the result
- y is a register holding the first number to add
- z is the second number to add

9.2 Description

Adds two numbers together. If x is not specified, then y becomes the destination.

9.3 Example

```
add r0, r1, #1
add r0, #1
```

10 sub

10.1 Syntax

```
sub x, y, z
sub y, z
```

Where:

- x is the destination for the result
- y is a register holding the first number to subtract
- z is the second number to subtract

10.2 Description

Subtracts z from y. If x is not specified, then y becomes the destination.

10.3 Example

11 rsb

11.1 Syntax

```
rsb x, y, z
```

Where:

- x is the destination register
- y is the register holding the first value
- z is the second value

11.2 Description

Just like the sub instruction it performs a subtraction on the two values. The difference however is that rsb will subtract y from z.

11.3 Example

```
rsb r0, r1, #10
rsb r0, #10
```

12 mul

12.1 Syntax

```
mul x, y, z
mul y, z
```

Where:

- x is the destination for the result
- y is a register holding the first number to multiply
- z is the second number to multiply

12.2 Description

Multiplies y and z. If x is not specified, then y becomes the destination.

12.3 Example

```
mul r0, r1, #2
mul r0, #2
```

13 b

13.1 Syntax

```
bx y
```

Where:

- x is the condition for branching
- y is the label to branch to

13.2 Description

Branch instruction that allows for jumping to labels in code.

13.3 Example

```
loop:
  ; do some stuff
b loop
```

14 push

14.1 Syntax

```
push x
push (x, y, z, ...)
```

Where:

- x, y, z, ... is the value to push onto the stack

14.2 Description

Allows for pushing of values from registers onto the stack.

14.3 Example

```
push #1
push (#1, #2, #3)
```

15 pop

15.1 Syntax

```
pop x
pop (x, y, z, ...)
```

Where:

- x, y, z, ... is the register to store the value popped off the stack

15.2 Description

Allows for popping of values off the stack into registers.

15.3 Example

```
pop r0
pop (r0, r1, r2)
```

16 lsl

16.1 Syntax

```
lsl x, y
```

Where:

- x is the register holding the value to shift
- y is the amount to shift the value by in decimal

16.2 Description

Logical shift left of a binary value.

16.3 Example

```
mov r1, #1
lsl r1, #24
```

17 lsr

17.1 Syntax

```
lsr x, y
```

Where:

- x is the register holding the value to shift
- y is the amount to shift the value by in decimal

17.2 Description

Logical shift right of a binary value.

17.3 Example

```
mov r1, $0000FF
lsr r1, #10
```

18 cmp

18.1 Syntax

```
cmp x, y
```

Where:

- x is the first value to compare
- y is the second value to compare

18.2 Description

Compares two values to allow for conditional checks. Stores the result in the APSR.

18.3 Example

```
cmp r0, #1
```

19 bic

19.1 Syntax

```
bic x, y, z
```

Where:

- x is the destination register
- y is the register holding the value
- z is the bitmask

19.2 Description

Performs a bitwise and not operation on y using z as a bitmask.

19.3 Example

```
bic r1, r1, #7
```

20 tst

20.1 Syntax

```
tst x, y
```

Where:

- x is the register holding the value to test
- y is the bitmask

20.2 Description

Performs a bitwise and operation on x using y as a bitmask. Stores test result in the APSR.

20.3 Example

```
tst r0, #1024
```