Digital Electronics and Microprocessor Systems (ELEC211)

Dave McIntosh and **Valerio Selis**

dmc@liverpool.ac.uk
V.Selis@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Outline

- Indirect addressing
- Endianness
  - Big Endian
  - Little Endian
- Endianness of the ARM Cortex M0
  - Little or Big Endian
  - Reversing byte order
- Loading bytes, half words and words (Loading instructions)
- Storing bytes, half words and words (Storing instructions)

Indirect Addressing

Indirect (or register indirect) addressing uses a value in a register to identify a memory address e.g. the instruction:

LDR r6, [r1]

means put (or ‘load’) into register r6 the data held in the memory location that has the address given by the value in register r1.

Load and Store

When a load instruction, LDR, is executed the data travels from memory to register

whereas when a store instruction, STR, is executed the data travels from register to memory.

Each memory location holds one byte or 8 bits of data whereas each register holds 4 bytes of data.

So LDR and STR use four consecutive memory addresses but **which byte goes to which location**?

Endianness

- What is endianness?
  - How we store bytes in the memory.
- Least significant bytes first.
  - Little Endian.
- Most significant bytes first.
  - Big Endian.

Little endian

Microprocessors can be either ‘little endian’ or ‘big endian’ - if the processor is ‘little endian’ then the instruction:

STR r6, [r1]

with 0xFFAABB11 in r6 and 0x00008000 in r1 would store:

byte	0x11	at address	0x00008000
	0xBB	at	0x00008001
	0xAA	at	0x00008002
	0xFF	at	0x00008003

Big endian

Whereas if the processor is ‘big endian’ then the instruction:

STR r6, [r1]

with 0xFFAABB11 in r6 and 0x00008000 in r1 would store:

byte	0xFF	at address	0x00008000
	0xAA	at	0x00008001
	0xBB	at	0x00008002
	0x11	at	0x00008003

Little endian or Big endian

For little endian the least significant byte (‘the little end’) is stored at the lowest memory address

whereas for big endian the most significant byte (‘the big end’) is stored at the lowest address.

The ARM processor can be configured as either little endian or big endian.

Both LDR and STR must use the correct endian configuration.

Question

When poll is active, respond at **PollEv.com/elec211**

Text **ELEC211** to **22333** once to join

0x4D, 0xA0, 0x94 and 0x0F are stored at memory addresses 0x00006000, 0x00006001, 0x00006002, 0x00006003 respectively.
What value is held in r0 after the execution of
LDR r0,[r7]
for (a) little endian and (b) big endian?

- little endian r0=0x4DA0940F, big endian r0=0x0F94A04D
- little endian r0=0x940FA04D, big endian r0=0x4DA0940F
- little endian r0=0x0F94A04D, big endian r0=0x4DA0940F
- little endian r0=0x4DA0940F, big endian r0=0x940FA04D

Total Results: 0

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

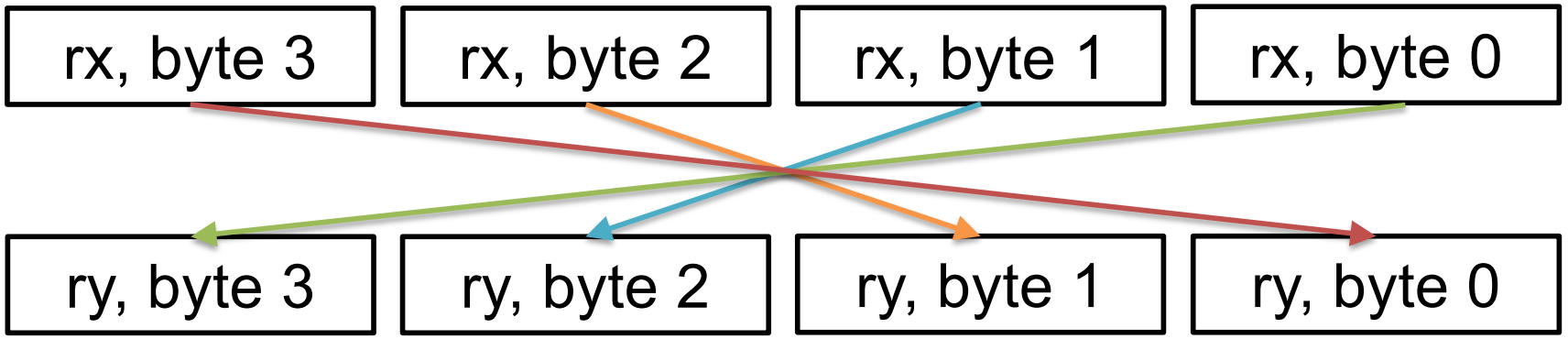
UNIVERSITY OF LIVERPOOL

Reversing byte order

The ARM Cortex M0 processor has an instruction that switches data in registers between big endian and little endian.

REV ry, rx

Bytes 0, 1, 2 and 3 of register rx are copied into bytes 3, 2, 1 and 0 of register ry in that order.



Reversing byte order

For example, the instruction

REV r0, r5

with **0xF0ACB714** in register r5 would put the value **0x14B7ACF0** into register r0.

The same instruction can convert little endian into big endian and big endian into little endian.

It is limited to the ‘low’ registers, r0 to r7, only.

Question

When poll is active, respond at **PollEv.com/elec211**

Text **ELEC211** to **22333** once to join

r1, r2 and r3 hold the values 0xFFBEADDE, 0xE5AFDCBA and 0xE5A55ED1 respectively.

What values are held in r4, r5 and r6 after the execution of the following?

REV r6,r2

REV r4,r1

REV r5,r3

- r4=0xBEFFDEAD, r5=0xA5E5D15E, r6=0xAFE5BADC
- r4=0xFFBEADDE, r5=0xE5A55ED1, r6=0xE5AFDCBA
- r4=0xBADCAFE5, r5=0xBADCAFE5, r6=0xBADCAFE5
- r4=0xDEADBEFF, r5=0xD15EA5E5, r6=0xBADCAFE5

Total Results: 0

Answers to this poll are anonymous

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

UNIVERSITY OF LIVERPOOL

Loading half words

The instruction LDR loads a register with a full word i.e. 4 bytes or 32 bits.

A different instruction, LDRH uses indirect addressing to identify 2 locations in memory to load a register with 2 bytes or 16 bits.

LDRH loads the lowest 16 bits of the register and the remaining 16 bits of the register are set to zero.

Another instruction, LDRSH, uses the sign of the half word to fill in the top 16 bits.

Example: half words

Assuming little endian and the base register, r6, holds the value 0x00004000.

If memory location with address 0x00004000 holds the byte 0x7B and the memory location with address 0x00004001 holds the byte 0x65 then:

Both LDRH r1, [r6] and LDRSH r1, [r6] would put 0x0000657B into register r1

0x657B = $25{,}979_{10}$ is a positive number in 16 bit 2's complement so sign bit is 0.

Example: half words

If memory location with address 0x00004000 holds the byte 0x7B and the memory location with address 0x00004001 holds the byte 0xA5 then:

LDRH r1, [r6] would put 0x0000A57B into r1

LDRSH r1, [r6] would put 0xFFFFA57B into r1

0xA57B = $-23{,}173_{10}$ is a negative number in 16 bit 2's complement so the sign bit is 1

1111 1111 1111 1111 1010 0101 0111 1011

Loading bytes

The instructions LDRB and LDRSB load the lowest 8 bits respectively of a register from a memory location given by indirect addressing.

The remaining 24 bits of the register are set to zero (for LDRB) or to the sign bit (for LDRSB).

Using LDRSB, then 0x000000ZZ is loaded into the destination register if byte 0xZZ is positive and 0xFFFFFFZZ is loaded if 0xZZ is negative.

Example: loading bytes

If register r6 holds the value 0x00004000 and memory location with address 0x00004000 holds the byte 0xA5 then:

LDRB r1, [r6] would put 0x000000A5 into register r1

LDRSB r1, [r6] would put 0xFFFFFFA5 into r1

0xA5 = -91_{10} is a negative number in 8 bit 2's complement so the sign bit is 1

1111 1111 1111 1111 1111 1111 1010 0101

Question

0x4D at 0x00006000
0xA0 at 0x00006001
0x94 at 0x00006002
0x0F at 0x00006003

r0 holds 0x00006000
r1 holds 0x00006002

LDRH r2, [r0]
LDRSH r4, [r1]
LDRSB r6, [r0]

LDRSH r3, [r0]
LDRB r5, [r1]
LDRSB r7, [r1]

When poll is active, respond at **PollEv.com/elec211**
Text **ELEC211** to **22333** once to join

What values are held in r2 - r7 after the execution of the instructions assuming little endian?

r2=0x00004DA0, r3=0xFFFF4DA0, r4=0x0000940F, r5=0x00009400, r6=0x00004D00, r7=0xFFFFFF49

r2=0x0000A04D, r3=0xFFFFA04D, r4=0x00000F94, r5=0x00000094, r6=0x0000004D, r7=0xFFFFFF94

r2=0xFFFFA04D, r3=0x0000A04D, r4=0x00000F94, r5=0x00000094, r6=0xFFFFFF4D, r7=0x00000094

r2=0x0000A04D, r3=0xFFFFA04D, r4=0xFFFF0F94, r5=0xFFFFFF94, r6=0x0000004D, r7=0xFFFFFF94

Total Results: 0

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

Storing half words and bytes

STRH stores the lowest half word or 16 bits of a register in 2 memory locations given by indirect addressing. (Endian rule applies.)

STRB stores the lowest byte or 8 bits of a register at a memory location given by indirect addressing. (Endian not important.)

Base plus offset addressing

Base plus offset addressing uses a value in a register (the ‘base’) plus a binary number (the ‘offset’) to identify a memory address.

E.g.

LDR r6, [r1, #12]

means load into r6 the data held in the memory location that has the address given by the value in register r1 added to the offset value 12.

Offset using another register

The offset can also be held in another register.

E.g.

LDR r6, [r1, r4]

means load into r6 the data held in the memory location that has the address given by the value in register r1 added to the ‘offset’ value held in register r4.

Words, half words and bytes

Base plus offset addressing can be used for both load and store, for words, half words and bytes including signed half words and signed bytes.

E.g.

```
STRH r6, [r1, r4]

LDRSB r2, [r3, #17]

LDRH r5, [r0, r7]
```

Words and half words alignment

The address value calculated by base plus offset addressing must be:

- ‘word aligned’ for LDR and STR
- ‘half word aligned’ for LDRH, LDRSH and STRH.

A ‘word aligned’ address is divisible by 4 and words are stored in 4 adjacent memory locations.

A ‘half word aligned’ address is divisible by 2 and half words are stored in 2 adjacent memory locations.

Offset immediates

The offset value used in base plus offset addressing are all 5 bits immediates.

For byte accesses (LDRB, LDRSB and STRB) the offset is in the range 0 to 31.

For half word accesses (LDRH, LDRSH and STRH) the offset is an even number in the range from 0 to 62.

For word accesses (LDR and STR) the offset is a number divisible by 4 in the range 0 to 124.

Addressing modes: a summary

Register addressing: machine code includes a number(s) that identifies a register(s) to be used by the instruction. The register contains the value.

Immediate addressing: machine code includes a number (the 'immediate') that is used directly by the instruction. Restrictions apply to the value that the immediate can take.

Addressing modes: a summary

Indirect addressing: the value to be used by the instruction is held in a memory location with an address given by the contents of a register referred to in the machine code.

Base plus offset addressing: similar to indirect addressing but the memory address is found by adding a value (the 'offset') to the contents of a base register. The offset can be either an immediate or held in another register.

Announcements

- **Homework 1** on the Microprocessor Systems lectures is due by 23:59 on the **20th February 2020**

Summary

- Endianness (Little endian and Big endian)
- Loading bytes, half words and words
- Storing bytes, half words and words
- Addressing modes (recap)

Next class?

Monday at 4 p.m. in the
Building 502,
Lecture Theatre 2
(502-LT2)