Memory Addressing, LDR and STR

A common workflow

- The CPU and ALU work with values stored locally in registers
- A standard workflow for operating on values in memory:
  - Load value(s) into registers
  - Perform operations on values in registers
  - Write result to register(s)
  - Store value(s) back to memory

LDR and STR

- LDR (‘Load Register’) copies a value from a memory location into a register
- STR (‘Store Register) copies a value from a register into a memory location
- E.g.:
  - `LDR R0,0xfc` loads the contents of the word starting at memory address 0x000fc into R0.
  - `STR R1,120` stores the contents of R1 into the word starting at memory address (decimal) 120.
  - `STR R3,myLabel` stores the contents of R3 into the word at the address represented by label `myLabel`.
  - `LDR R4, myLabel` loads the contents of the word at the address represented by label `myLabel`

LDR and STR (cont)

- The address passed to LDR/STR must be ‘word aligned’
  - that is, it must be divisible by 4
- Therefore:
  - `LDR R3,0x81` is an invalid instruction because address 0x00081 is not a valid word address (it is not divisible by four).
  - The Assembler will trap this.

LDR and STR (cont)

- The address passed to LDR/STR must be ‘word aligned’
  - that is, it must be divisible by 4
- Therefore:
  - `LDR R3,0x81` is an invalid instruction because address 0x00081 is not a valid word address (it is not divisible by four).
  - The Assembler will trap this.
- But wait ! You said every byte is addressable didn’t you ?

LDRB and STRB

- LDR and STR have byte addressable variants!
- *LDRB will load just a single byte of memory*
  - into the least significant 8 bits of a register – setting the other bits to 0
- *STRB will store the least significant 8-bits of a register into a specified byte of memory*
  - not altering any neighbouring bytes

LDRB and STRB (cont)

- LDRB and STRB do not require the specified memory address to be word aligned (divisible by four).
- For example
  - `LDRB R3,0x81` is a valid instruction, and will load the contents of the single byte at address 0x00081 into the least significant 8 bits of R3.

Indirect Addressing

- LDR and STR also support ‘indirect addressing’
- Indirect addressing allows memory to be addressed via registers holding the address
- In some programming languages (e.g., C), this is the basis of ‘pointers’ – a memory word that holds the address of another memory word
- In ARM assembly:
  - `LDR R0,[R1]` will load into R0, the contents of the memory address that is currently held in R1.
  - `STR R3,[R4]` will store the value in R3 to the memory word with address currently held in R4
- We’ll come back to indirect addressing next week !