

# COMP 122/L Lecture 12

Mahdi Ebrahimi

Slides adapted from Dr. Kyle Dewey

# Outline

- Memory instructions
  - Load (`ldr`)
  - Store (`str`)
- Arrays

# Memory Operations

# Refresher

You've already seen one form of `ldr` for handling strings

# Refresher

You've already seen one form of `ldr` for handling strings

---

```
.data
my_string:
    .asciz "hello"

.text
ldr r0, =my_string
```

# Refresher

You've already seen one form of `ldr` for handling strings

---

```
.data
my_string:
    .asciz "hello"

.text
ldr r0, =my_string
```

**Puts starting address of "hello" in r0**

- This was used for setting up swi instructions
- While this uses memory, it did so indirectly (the swi instruction actually read the memory, not us)

# Putting Integers in Memory

- `.word` directive will put a 32-bit integer in memory, much like `.asciz` will put a string in memory

# Putting Integers in Memory

- `.word` directive will put a 32-bit integer in memory, much like `.asciz` will put a string in memory

```
.data
my_string:
    .asciz "hello"
first_int:
    .word 42
second_int:
    .word 38
```



# Reading Integers From Memory

Step 1: use `ldr` to put its address into a register...

# Reading Integers From Memory

Step 1: use `ldr` to put its address into a register..

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```

# Reading Integers From Memory

Step 2: use `ldr` with `[]` to read the value at the address

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```

# Reading Integers From Memory

Step 2: use `ldr` with `[]` to read the value at the address

```
.data
first_int:
    .word 42
second_int:
    .word 38

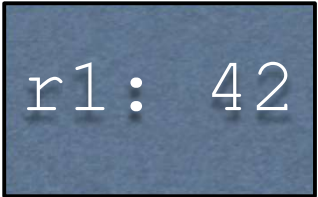
.text
ldr r0, =first_int
ldr r1, [r0]
```

# Reading Integers From Memory

Step 2: use `ldr` with `[]` to read the value at the address

```
.data
first_int:
    .word 42
second_int:
    .word 38
```

```
.text
ldr r0, =first_int
ldr r1, [r0]
```



r1: 42

# Writing Integers to Memory

Step 1: use `ldr` to put its address into a register...

# Writing Integers to Memory

Step 1: use `ldr` to put its address into a register..

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```

# Writing Integers to Memory

Step 2: use `str` to write a value at that address

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```



# Writing Integers to Memory

Step 2: use `str` to write a value at that address

```
.data
first_int:
    .word 42 57
second_int:
    .word 38

.text
ldr r0, =first_int
mov r1, #57
str r1, [r0]
```

# Example:

`memory_variables.s`

# Arrays

# Specifying Arrays

Only distinction from variables:  
multiple values are specified with the `.word` directive

# Specifying Arrays

Only distinction from variables:  
multiple values are specified with the `.word` directive

---

```
.data  
first_int:  
    .word 42
```

# Specifying Arrays

Only distinction from variables:  
multiple values are specified with the `.word` directive

---

```
.data
first_int:
    .word 42
array:
    .word 32, 65, 76, 87
```

# Accessing Arrays

Basic approach: increment memory address

# Accessing Arrays

Basic approach: increment memory address

---

```
.data
arr:
    .word 32, 65, 76
.text
ldr r0, =arr
ldr r1, [r0]
add r0, r0, #4
ldr r2, [r0]
add r0, r0, #4
ldr r3, [r0]
```


-Offsets increment by 4 because one word is 4 bytes



# Accessing Arrays

Basic approach: increment memory address

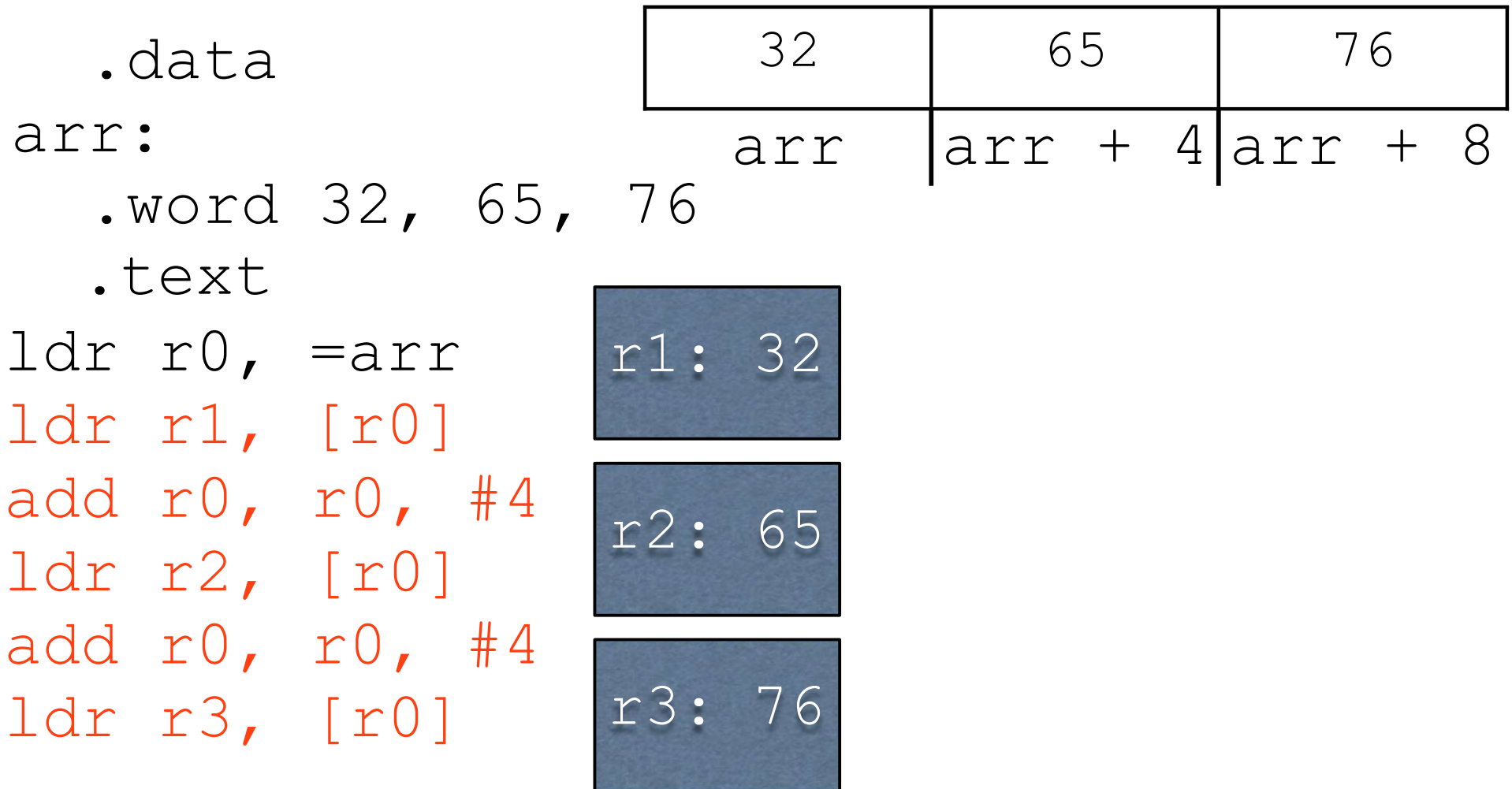
```
.data
arr:
    .word 32, 65, 76
.text
ldr r0, =arr
ldr r1, [r0]
add r0, r0, #4
ldr r2, [r0]
add r0, r0, #4
ldr r3, [r0]
```



r1: 32
r2: 65
r3: 76

# Accessing Arrays

Basic approach: increment memory address



-Top right corner shows memory layout in terms of arr

# Example:

```
register_indirect.s
```

# More on Memory Access

```
ldr r3, [r0]
```

- The above instruction uses the *register indirect addressing mode*
  - Addressing mode: how the processor accesses something
  - Register indirect: Memory access is done through an address in a register
- Many more available: see `register_*.s`

# ARM addressing Modes

Name	Alternative Name	ARM Examples
Register to register	Register direct	MOV R0, R1
Absolute	Direct	LDR R0, MEM
Literal	Immediate	MOV R0, #15 ADD R1, R2, #12
Indexed, base	Register indirect	LDR R0, [R1]
Pre-indexed, base with displacement	Register indirect with offset	LDR R0, [R1, #4]
Pre-indexed, autoindexing	Register indirect pre-incrementing	LDR R0, [R1, #4]!
Post-indexing, autoindexed	Register indirect post-increment	LDR R0, [R1], #4
Double Reg indirect	Register indirect Register indexed	LDR R0, [R1, R2]
Double Reg indirect with scaling	Register indirect indexed with scaling	LDR R0, [R1, r2, LSL #2]
Program counter relative		LDR R0, [PC, #offset]

# ARM addressing Modes

Addressing Mode	Assembly Mnemonic	Effective address	FinalValue in R1
Indexed, base Register indirect	LDR R0, [R1]	R1	R1
Pre-indexed, base with displacement	LDR R0, [R1, #d]	$R1 + d$	R1
Pre-indexed, autoindexing	LDR R0, [R1, #d]!	$R1 + d$	$R1 + d$
Post-indexed, autoindexed	LDR R0, [R1], #d	R1	$R1 + d$
Double Reg indirect	LDR R0, [R1, R2]	$R1 + R2$	R1
Double Reg indirect with scaling	LDR R0, [R1, R2, LSL #2]	$R1 + (R2 * 4)$	R1 (R2 also unchanged)

# Array Access Example:

```
print_array_fixed_length.s
```

# Writing to Array Example:

```
write_array_increasing.s
```



## Another Array Access Example:

```
print_array_variable_length.s
```