HW 4.2

a → r0 , b → r1 , c → r2 , d → r3

```
do {
    b += 0;
    c += 1;
} while (c <= 1);
```

```
.Lstart_c:
    add r1, r1, 0
    add r2, r2, 1
    cmp r2, 1
    ble .Lstart_c
```

HW 4.3 For Loop

```
    mov r0, 6
.Lstart_a:
    cmp r0, 2
    blt .Lend_a
    add r1, r1, 7
    sub r0, r0, 2
    b .Lstart_a
.Lend_a:
```

```
for ( a = 6 ; a >= 2 ;
      a -= 2 )
{
    b += 7;
}
```

HW 4.4 For loop (C to ARM)

```
for (b=2; b<=7; b+=1)
    { a+=1;
    }
```

```
mov r1,2
.Lstart_b:
  cmp r1,7
  bgt .Lend_b
  add r0,r0,1
  add r1,r1,1
  b .Lstart_b
.Lend_b
```

HW 4.5 ARM Pointer Arithmetic

size of (char) = 1

size of (short) = 2

size of (int) = 4

size of (double) = 8

arr → r4

p → r0

double *p = arr + 6

num

double → num × 8

int → num × 4

short → num × 2

char → num × 1

6 × 8 = 48

add r0, r4, 48

HW 4.6 Endianness

Little endian 反过来 "⟵"

Big endian 正 "⟶"

HW4.6. Endianness

Assume we are working with a little-endian ISA and have the following memory addresses and values.

address	100	101	102	103	104	105	106	107	108	109
value	0x12	0x5D	0x94	0x6A	0xA2	0x5D	0x54	0x56	0xC1	0x61

After executing the code below, what will the value of `r1` be?

```
mov r4, 100
ldr r1, [r4, 0]
```

r4 = 100

⇒ 6A945D12

看清楚

ldr 4 byte, 4 hex

ldrh 2 byte

ldrb 1 byte

Hw 4.7 Array Access C to ARM

r0 = &B[2] → address

r2 = &Y

B short

Y short → 2 bytes

load address → value

Y = B[8];

ldrh

⇓

ldrh r1, [r0, off]

strh r1, [r2]

off = 8 × 2 = 16

每個 address 存 "1 byte"

Hw 4.8
pointers ARM to C

HW4.8. Pointers ARM to C

r0 = &x r1 = &y

Suppose the initial value in `r0` is the address of `x`, i.e. `r0 = &x`. The initial value in `r1` is the address of `y`, i.e. `r1 = &y`.

Assume that we have variables `x` and `y` correctly defined for each subproblem so that the corresponding C statement makes sen
to write (you can ignore the types).

Determine which of the given C statements correctly implements the following ARM assembly

```
ldr r2, [r0]
ldr r2, [r2]
ldr r3, [r1]
ldr r3, [r3]
str r2, [r3]
```

r2 = x
r2 = *x
r3 = y
r3 = *y

*y = *x

variant:

ldr r2, [r0] r2 = x
ldr r2, [r2] r2 = *x
ldr r3, [r2] r3 = **x
str r2, [r1] y = **x

HW 4.9

先assign再++

→ y = x ++

先++再assign

y = ++x

HW4.9. Pointers ARM to C

Suppose `r0 = &x` and `r1 = &y`.

Assume that we have variables `x` and `y` correctly defined for each subproblem so that the corresponding C statement makes sense to write (you can ignore the types).

Determine which of the given C statements correctly implements the following ARM assembly

```
ldr r2, [r1]
ldr r2, [r2]
ldr r3, [r2]
add r3, r3, 1
str r3, [r2]
str r3, [r0]
```

r2 = y
r2 = *y
r3 = **y
++ 再store

x = ++(**y)

**y = **y + 1

r3 → ++(**y)

variant:

ldr r2, [r1] r2 = y
ldr r3, [r2] r3 = *y
str r3, [r0] x = *y
add r3, r3, 1 (*y)++
str r3, [r2] x = (*y)++

first assign

x = *y → r1

*y = *y + 1

then ++

→ (**y)++ → r2

x = (**y)++

4.10

HW4.10. For loop load array

Fill in the blanks in the ARM assembly and the corresponding C Code. Assume that `a` maps to `r0`, `b` maps to `r1`. Array `K` is an integer array with base address in `r7`.

K base address r7

```
for (a = 4; a <= 9; a += 3) {
    b  += K[a];
}
```

```
    mov r0, 4
    mov r5, 4
.Lstart_a:
    cmp r0, 9
    bgt .Lend_a
    mul r6, r0, r5
    ldr r8, [__blank0__, __blank1__]
    add __blank2__, r1, r8
    add r0, r0, 3
    b .Lstart_a
.Lend a:
```

r7 r6 r1

4.11 Load

$x = array[2][3]$

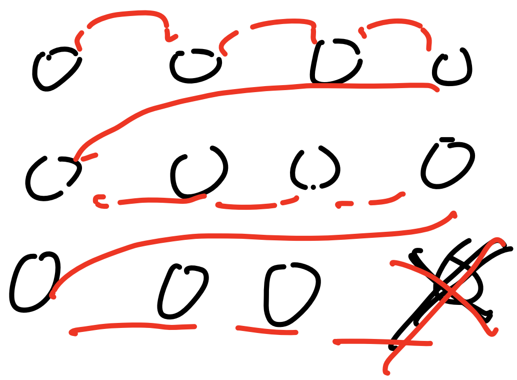
$array[3][4]$

int array

base address of array → r0

x → r4

row, column



+11 次 4

44

需要读出 44f =

```
mov r1, 2
lsl r3, r1, 2
mov r2, 3
add r3, r3, r2
```

$r_1 = 2$

$r_1 = 2 * 2^2 = 8$

$r_2 = 3$

$r_3 = 3 + 8 = 11$

lsl r3, r3, 2

ldr r4, [r0, r3]

$r_3 = 11 \times 2^2 = 44$

$x = (arr + 44)$