F28HS2 Hardware-Software Interface

Lecture 8: ARM Assembly Language 3

Array space

- array is continuous sequence of bytes

type name`[`*size*`]`

- allocate `sizeof(`*type*`)*`*size* bytes
- start register at address of 1st byte
- access byte/half word/word indirect on register
- add 1/2/4 to register to get to address for next byte/halfword/word

Array access

- `a[i]` == address for `a + i*size(`*type*`)`

- for iterative access to `a[i]`
  - `0<= i<`*size*
- put address of array in register `Ri`
- on each iteration
  - access `[Ri]`
  - add `size(`*type*`)` to `Ri`

Allocating memory

label `.rept` *count*

directives

`.endr`

- repeat *directives count* times
- use to allocate space e.g.

```
a:  .rept 10
    .byte 0x00
    .endr
```

- `a` is address of first of 10 bytes all set to `0`

Example: A[i] = i

```
#define MAX 25
int A[MAX];
int i;
i = 0;
while(i<MAX)
{   A[i] = i;
    i = i+1;
}
```

```
...
.data
.equ MAX, 25
A:  .rept MAX
    .word 0
    .endr
```

Example: A[i] = i

```
#define MAX 25
int A[MAX];
int i;
i = 0;
while(i<MAX)
{   A[i] = i;
    i = i+1;
}
```

```
.global _start
_start:
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
_exit:  MOV R0, #0
        MOV R7, #1
        SWI 0
```

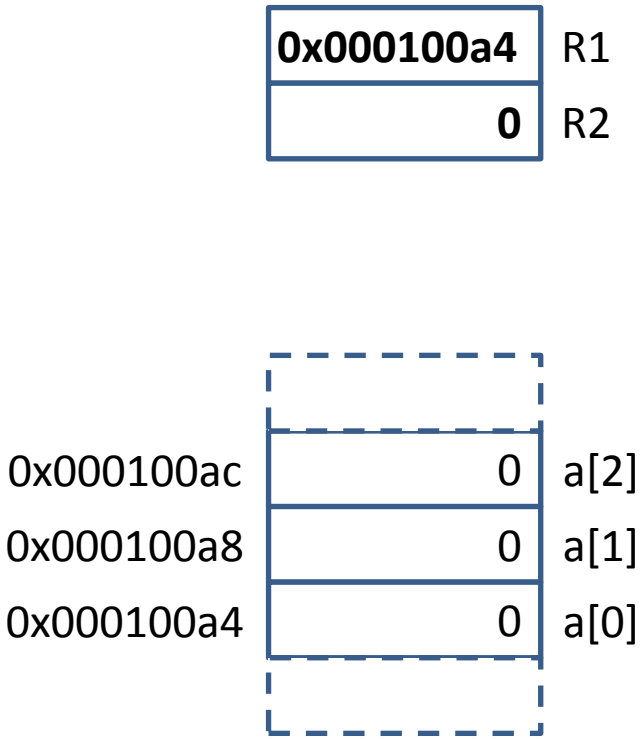
Example: A[i] = i

```
#define MAX 25
int A[MAX];
int i;
i = 0;
while(i<MAX)
{   A[i] = i;
    i = i+1;
}
```

```
(gdb) i variables
...
0x000100a4  A
...
(gdb) x/25w 0x100a4
0x100a4 <A>:     0    1    2    3
0x100b4 <A+16>:  4    5    6    7
0x100c4 <A+32>:  8    9   10   11
0x100d4 <A+48>: 12   13   14   15
0x100e4 <A+64>: 16   17   18   19
0x100f4 <A+80>: 20   21   22   23
0x10104 <A+96>: 24
(gdb)
```

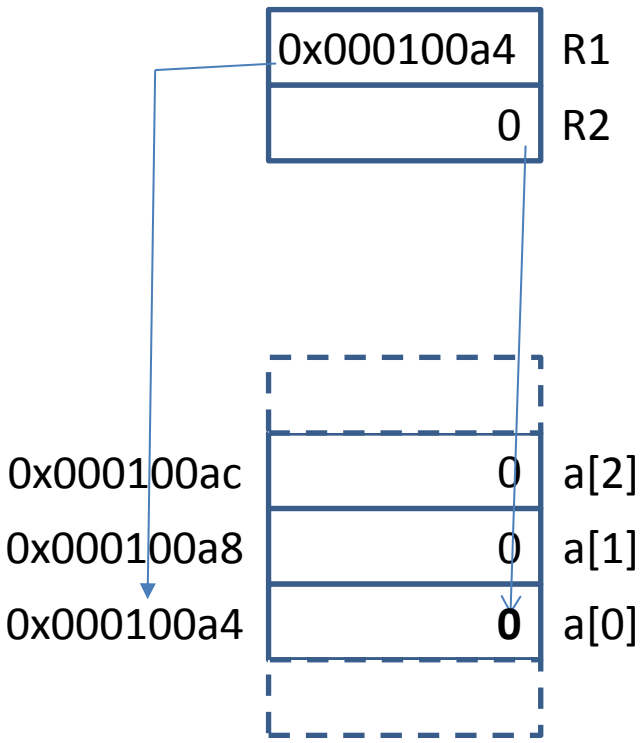
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



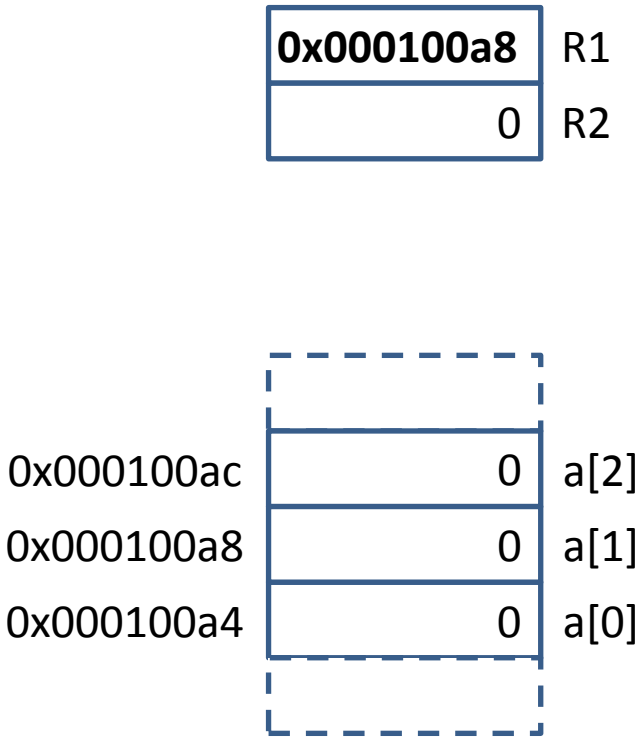
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



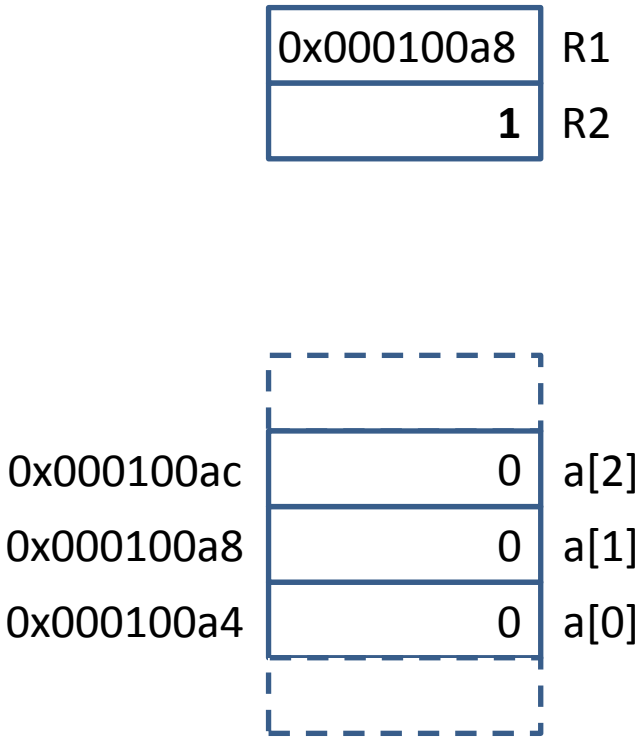
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



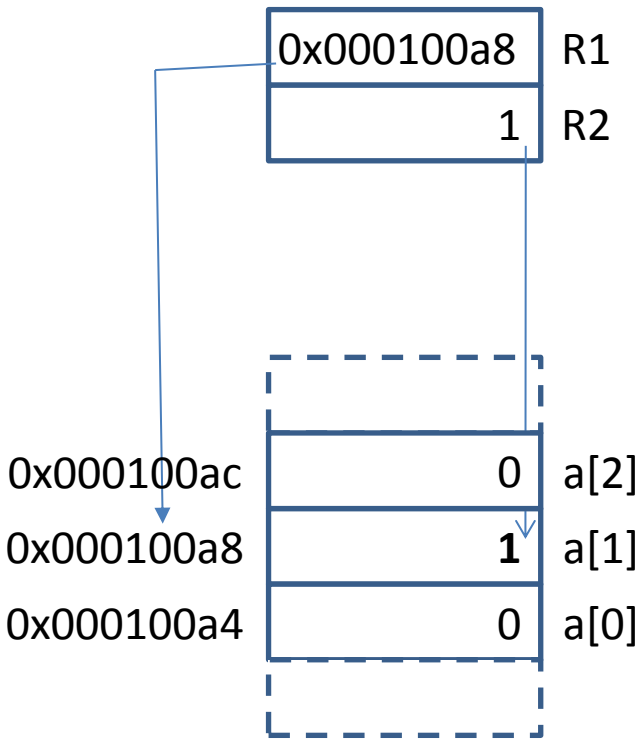
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



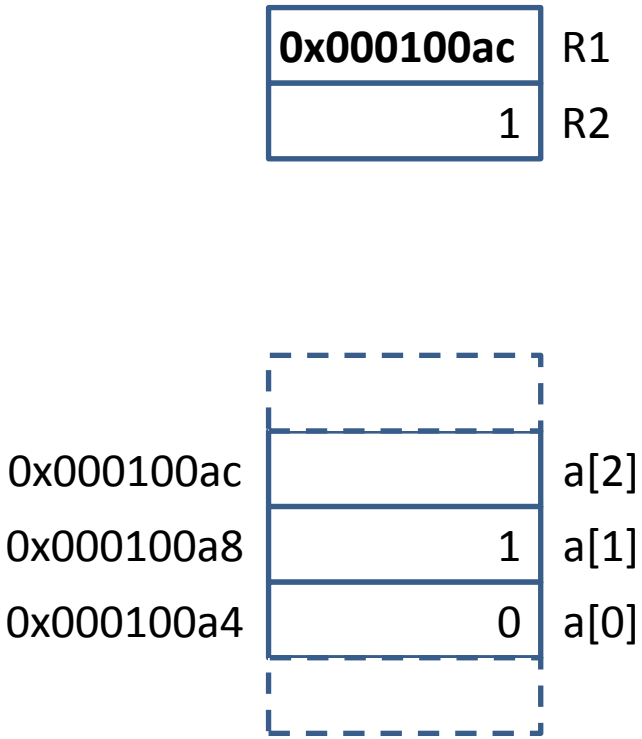
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



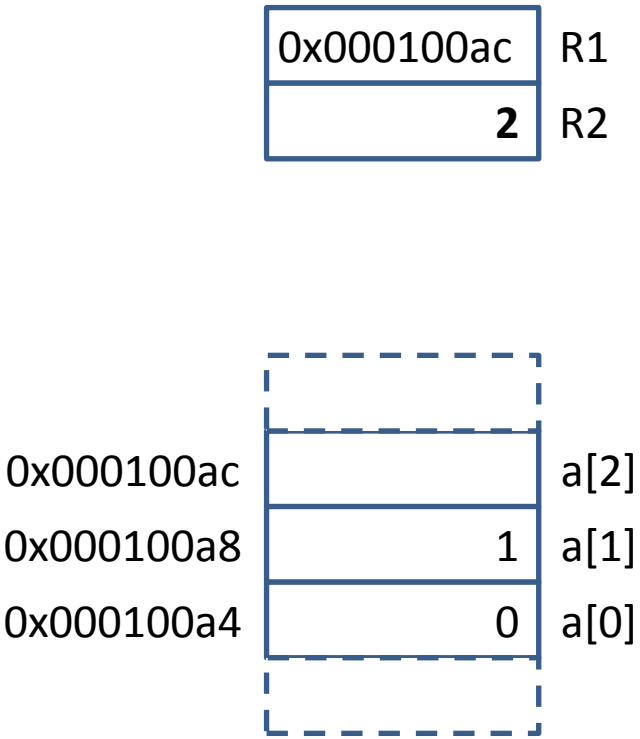
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



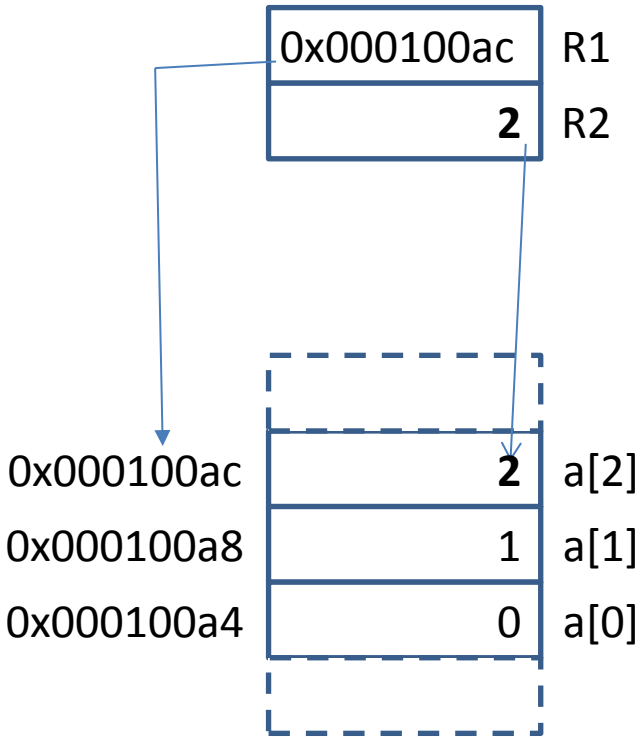
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



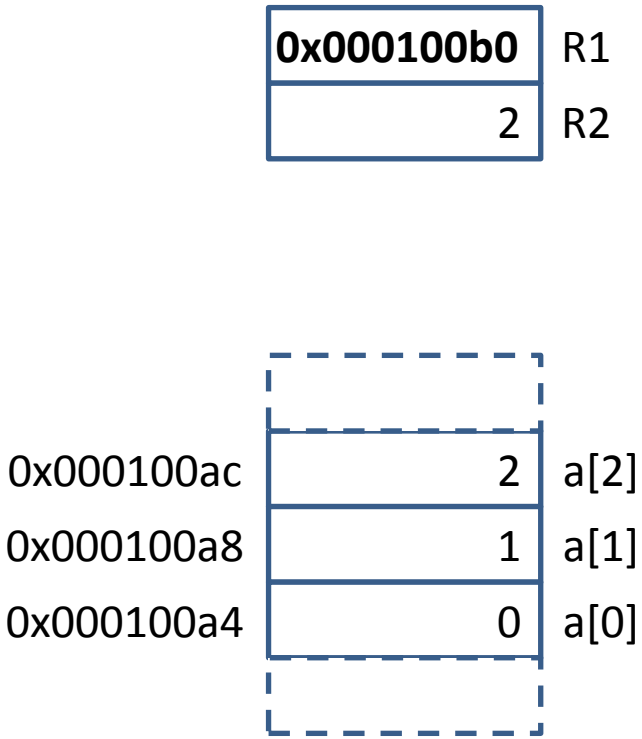
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



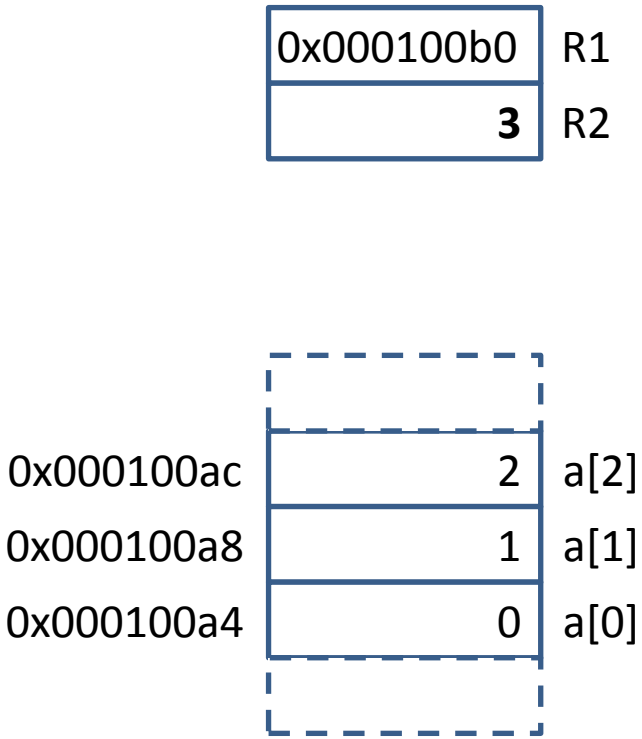
Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



Example: A[i] = i

```
        LDR R1,=A
        MOV R2,#0x00
_test:  CMP R2,#MAX
        BEQ _exit
        STR R2,[R1]
        ADD R1,#0x04
        ADD R2,#0x1
        B _test
```



Example - binary search

```
#define MAX 25
int A[MAX];
int l,r,m,v;
l = 0;
r = MAX-1;
while(l<=r)
{  m = (l+r)/2;
   if(A[m]==v)
    break;
   if(A[m]>v)
    r = m-1;
   else
    l = m+1;
}
```

Divide?

- `(l+r)/2`?
- use logical right shift!
- e.g. 7/2 == 3; 0111 >> 1 == 0011

`LSR` ***Rd, Rn, int***

- *Rd* = *Rn* >> *int*

Example - binary search

```
#define MAX 25
int A[MAX];
int l,r,m,v;
l = 0;
r = MAX-1;
while(l<=r)
{  m = (l+r)/2;
   if(A[m]==v)
    break;
   if(A[m]>v)
    r = m-1;
   else
    l = m+1;
}
```

```
.global _start
_start:
        ...
_search:
      MOV R6, #4       @ R6 == 4
      MOV R1, #0       @ R1 == l
      MOV R2, #MAX     @ R2 == r
      SUB R2, #1       @ R2 = MAX-1
```

Example - binary search

```
#define MAX 25
int A[MAX];
int l,r,m,v;
l = 0;
r = MAX-1;
while(l<=r)
{  m = (l+r)/2;
   if(A[m]==v)
    break;
   if(A[m]>v)
    r = m-1;
   else
    l = m+1;
}
```

```
_loop:
        CMP R1, R2      @ l>r?
        BGT _fail
        ADD R3, R1, R2  @ R3 == m
        LSR R3, R3, #1  @ m = (l+r)/2
        MUL R7, R3, R6  @ R7 = m*4
        LDR R4, =A      @ R4 == &A
        ADD R4, R7      @ R4 == &A[m]
        LDR R5, [R4]    @ R5 == A[m]
        CMP R5, #V      @ V==A[m]?
        BEQ _exit
```

Example - binary search

```
#define MAX 25
int A[MAX];
int l,r,m,v;
l = 0;
r = MAX-1;
while(l<=r)
{  m = (l+r)/2;
   if(A[m]==v)
    break;
   if(A[m]>v)
    r = m-1;
   else
    l = m+1;
}
```

```
        BGT _left
        MOV R1, R3      @ A[m]>V
        ADD R1, #1      @ l = m+1
        B _loop
_left:                  @ A[m]<V
        MOV R2, R3
        SUB R2, #1      @ r = m-1
        B _loop
```

Example - binary search

```
#define MAX 25
int A[MAX];
int l,r,m,v;
l = 0;
r = MAX-1;
while(l<=r)
{  m = (l+r)/2;
   if(A[m]==v)
    break;
   if(A[m]>v)
    r = m-1;
   else
    l = m+1;
}
```

```
_fail:
        MOV R3, #MAX
_exit:
        MOV R0, R3
        MOV R7, #1
        SWI 0
.data
.equ MAX, 25
A: .rept MAX
   .word 0
   .endr
.equ V, MAX-1
```

Where's my code?

- in `gdb` can *disassemble* memory areas to show code & addresses

`disassemble` *first*`,` *last*

- *first* & *last* can be labels or addresses

Where's my code?

```
_start:
        MOV R1, #X
        MOV R2, #Y
        MOV R3, #0
_loop:
        CMP R2, #0x00
        BEQ _exit
        ADD R3, R1
        SUB R2, #0x01
        B _loop
_exit:
        MOV R0, R3
        MOV R7, #1
        SWI 0
```

```
(gdb) disassemble _start,_exit
Dump of assembler code from 0x8054 to
0x8074:
0x8054 <_start+0>: mov r1, #3
0x8058 <_start+4>: mov r2, #4
0x805c <_start+8>: mov r3, #0
0x8060 <_loop+0>:  cmp r2, #0
0x8064 <_loop+4>:  beq 0x8074 <_exit>
0x8068 <_loop+8>:  add r3, r3, r1
0x806c <_loop+12>: sub r2, r2, #1
0x8070 <_loop+16>: b 0x8060 <_loop>
End of assembler dump.
(gdb)
```

Logical Shift

`LSL` *Rd, operand*

== *Rd* = *Rd* << *operand*

- add *operand* 0s to right
- like multiplying by $2^{operand}$

`LSR` *Rd, operand*

== *Rd* = *Rd* >> *operand*

- add *operand* 0s to left
- like dividing by $2^{operand}$

Logical operations

AND *Rd, Rn, operand*$_2$

== *Rd* = *Rn* & *operand2*

ORR *Rd, Rn, operand*$_2$

== *Rd* = *Rn* | *operand2*

EOR *Rd, Rn, operand*$_2$

== *Rd* = *Rn* ^ *operand2*

Strings

- in the `.data` section:

`.ascii` "*letters*"

  - bytes containing letters
  - 4 to a word
  - not 0 terminated

`.asciz` "*letters*"

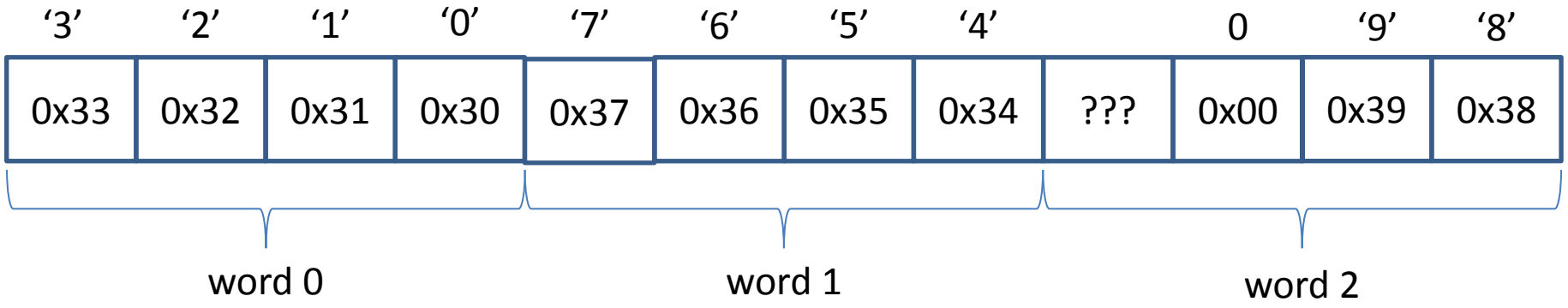
  - 0 terminated

- NB 4 bytes to a 32 bit word in "reverse order"

Strings

e.g.

```
digits: .asciz "0123456789"
```



String length

- count = 0
- get 1st word's address
- repeat
  - get word from address
  - bytes = 4
  - while bytes!=0 do
    - AND 1st word with 0xFF to get next byte
    - if next byte==0 then

      done
    - count = count + 1
    - shift word 8 bits right
    - bytes = bytes-1
  - move to next word address

String length

R1 == address of next word

R2 == count

R3 == contents of next word

R4 == next byte

R5 == byte count

```
.equ MASK, 0x000000ff
.equ EOS, 0
...
     LDR R1, =digits
     BL length
...
```

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
```

String length

R1 == address of next word

R2 == count

R3 == contents of next word

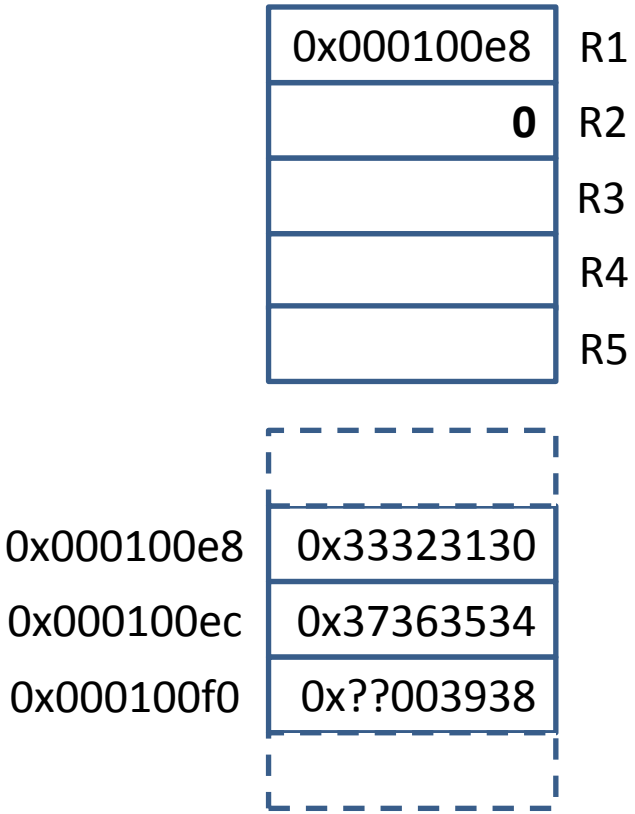
R4 == next byte

R5 == byte count

```
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```

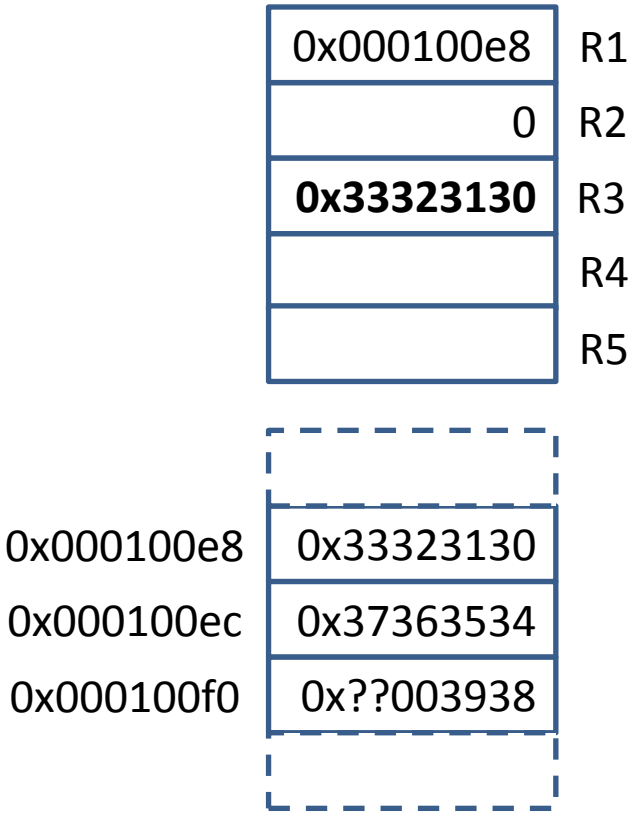
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



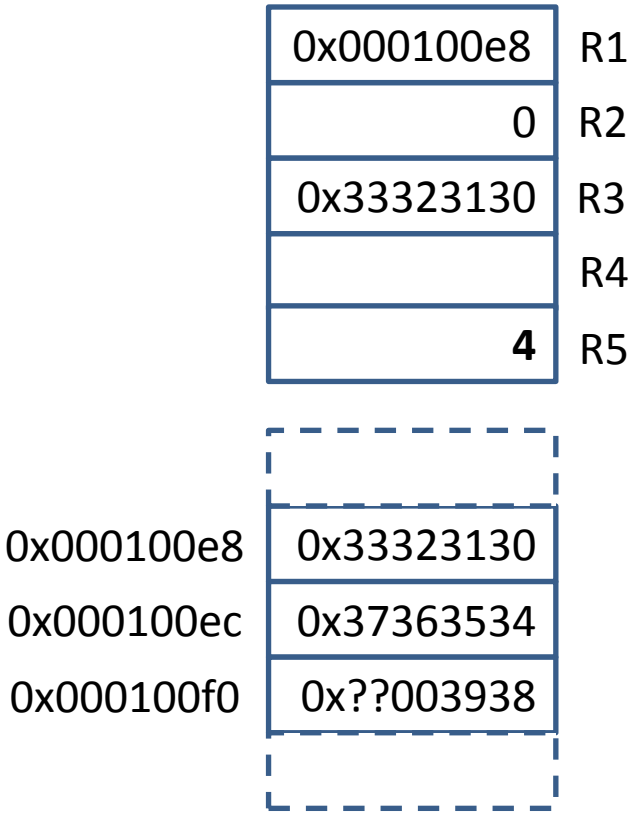
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



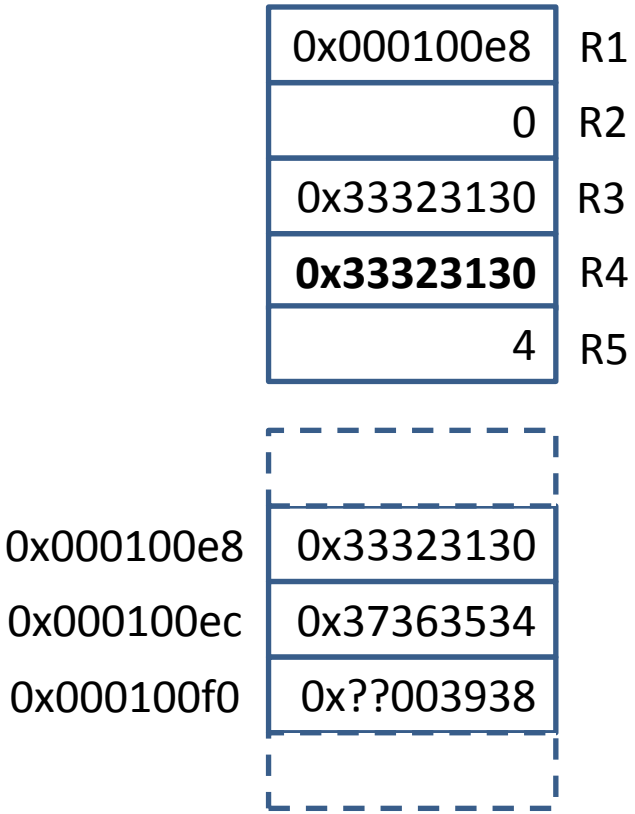
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



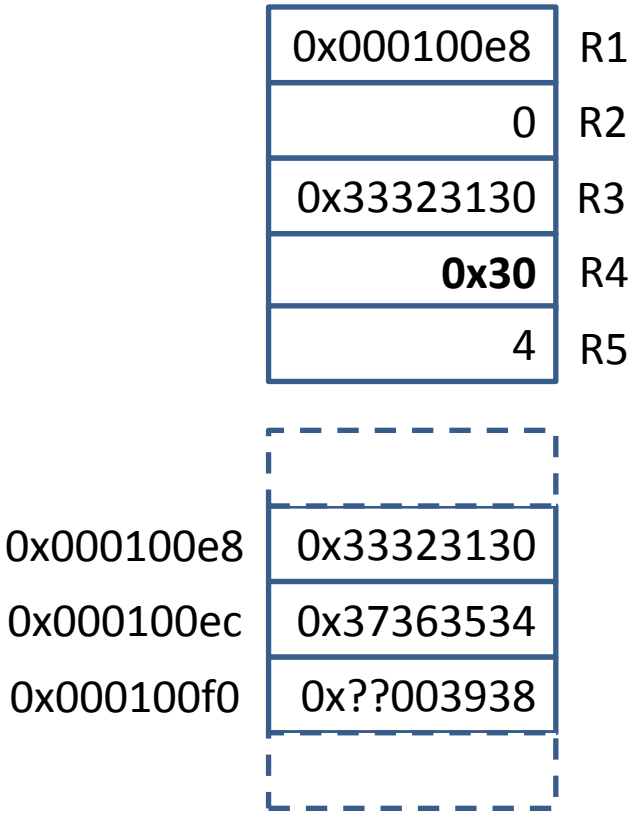
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



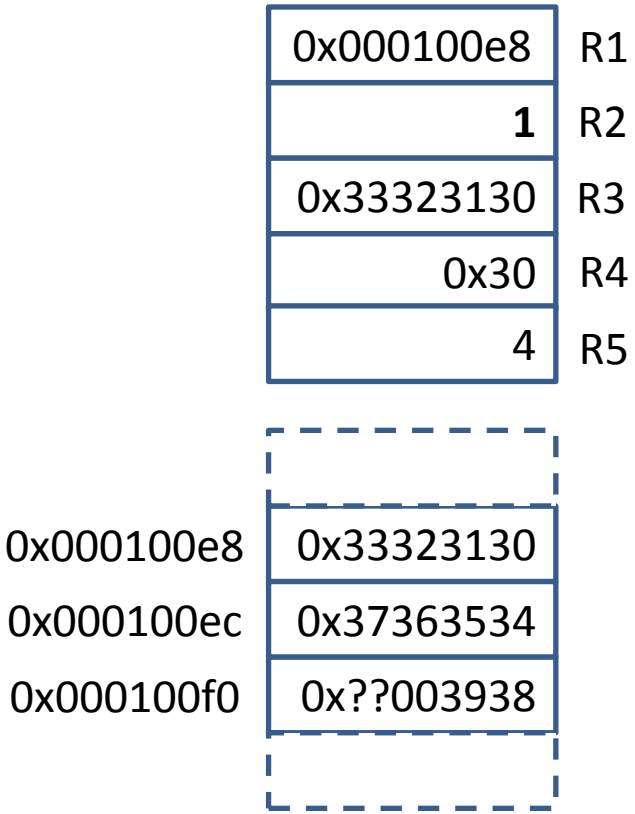
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



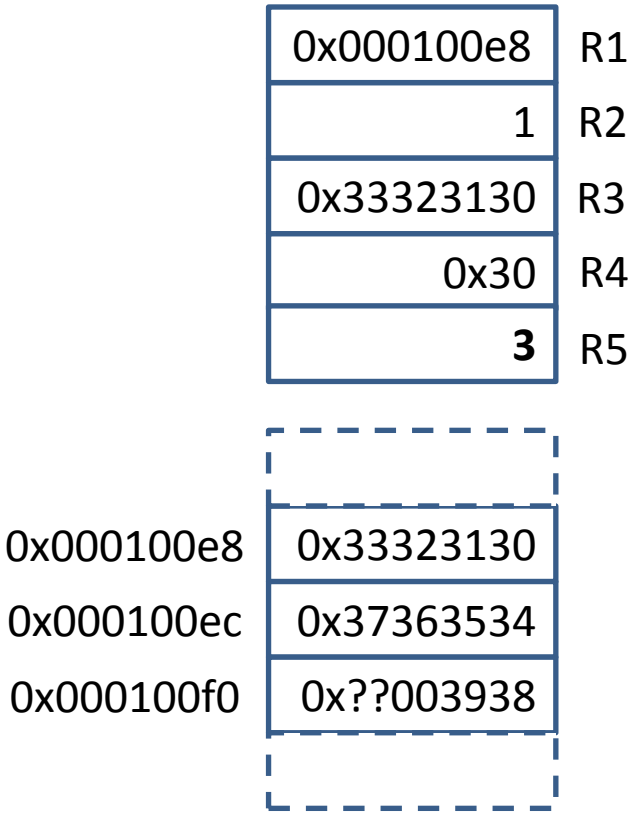
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



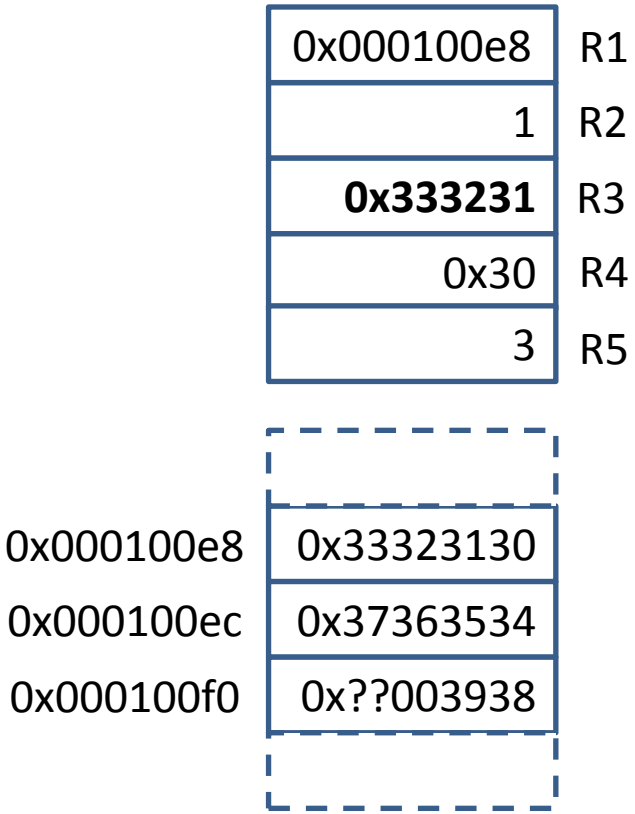
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



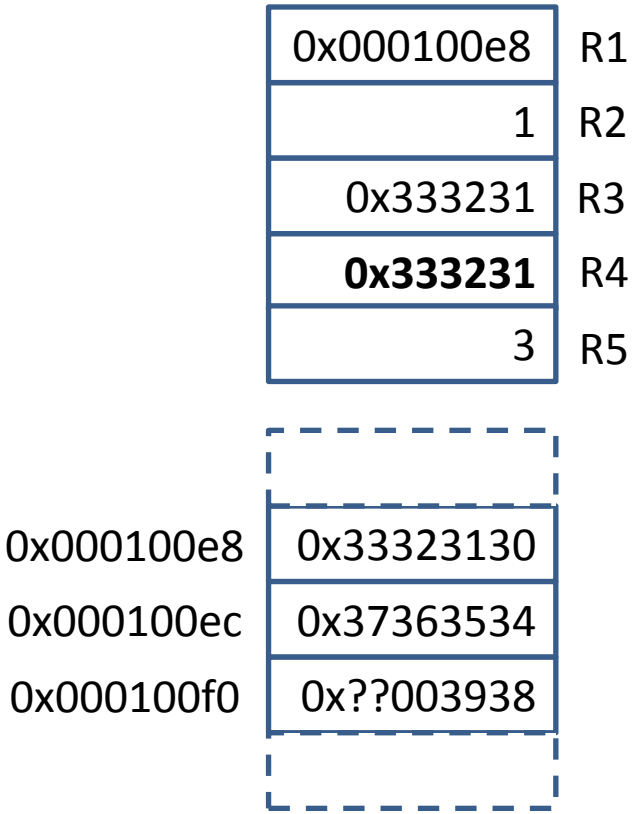
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



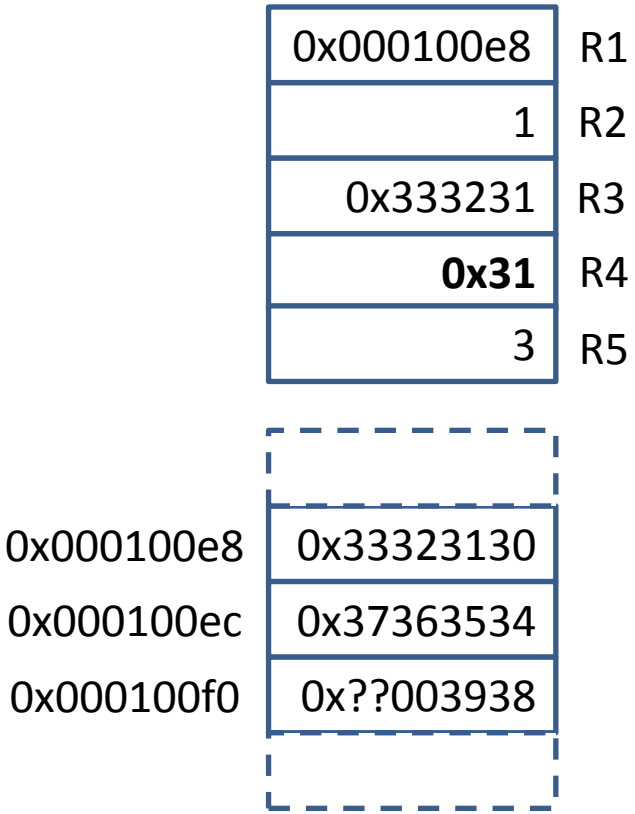
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```

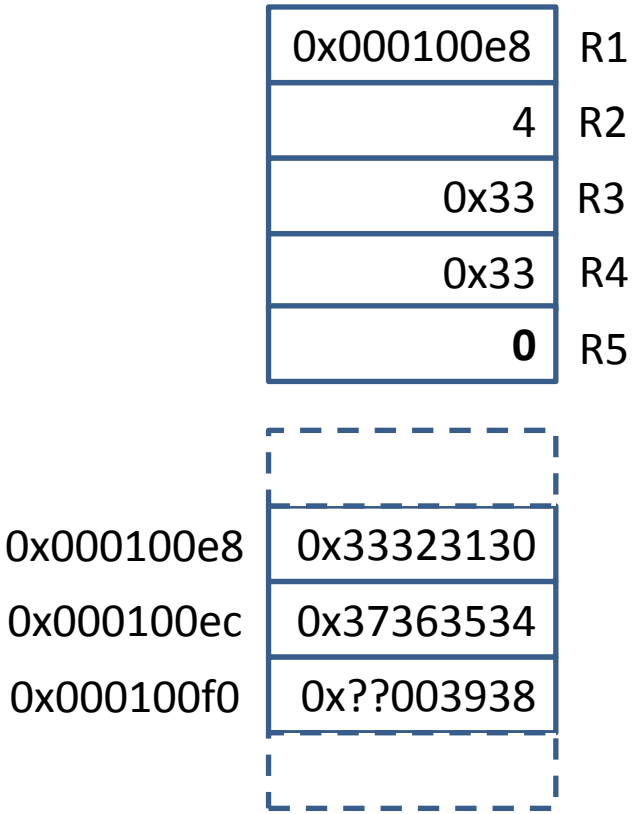


String length

- and so on...

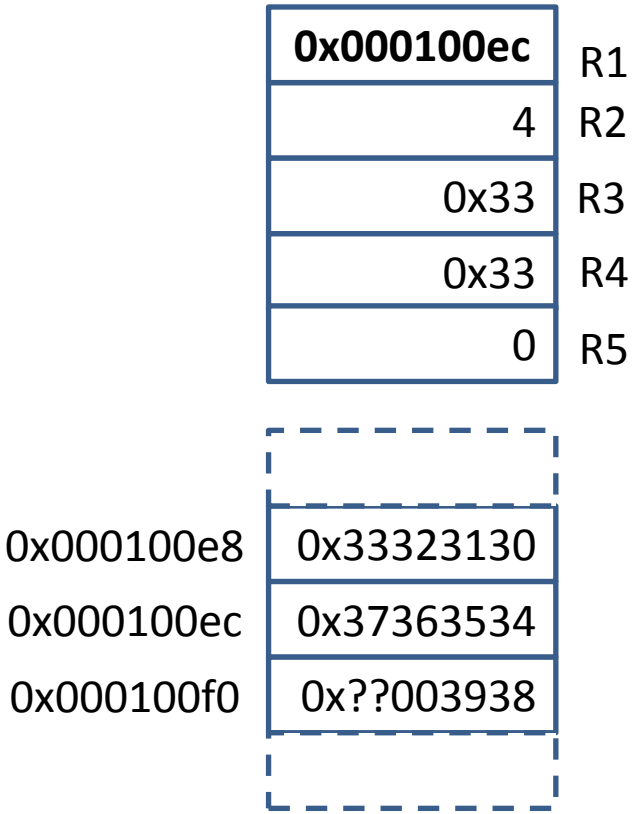
String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```



String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
       BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
       B nextw
endl:   BX LR
```



String length

```
length: MOV R2,#0
nextw:  LDR R3,[R1]
        MOV R5, #4
nextb:  MOV R4, R3
        AND R4, #MASK
        CMP R4, #EOS
        BEQ endl
        ADD R2, #1
        SUB R5, #1
        CMP R5, #0
        BEQ incr
        LSR R3, #8
        B nextb
incr:   ADD R1,#4
        B nextw
endl:   BX LR
```