F28HS2 Hardware-Software Interface

Lecture 7: ARM Assembly Language 2

Structured programming

- assembly language requires intricate use of labels & branches
- easy to produce "spaghetti code"
- design assembly programs using high level program structures
  - condition
  - iteration
- use template to translate high level to label + branch

Structured programming: while

WHILE exp_1 *op* exp_2 DO
command

```
_WLOOP:  Ri = exp1
         Rj = exp2
         CMP Ri,Rj
         Bnot(op) _WEND
         command
         B _WLOOP
_WEND:
```

Example: division

- count how often can take `y` from `x`

```
int x;
int y;
int q;
x = 23;
y = 2;
q = 0;
while(x>=y)
{ x = x-y;
  q = q+1;
}
```

```
 .global _start

_start:
        MOV R1, #X
        MOV R2, #Y
        MOV R3, #0 @ Q
_loop:
        CMP R1, R2
        BLT _exit
        SUB R1, R2
        ADD R3, #1
        B _loop
```

Example: division

- count how often can take `y` from `x`

```
int x;
int y;
int q;
x = 23;
y = 2;
q = 0;
while(x>=y)
{ x = x-y;
  q = q+1;
}
```

```
_exit:
        MOV R0, R3
        MOV R7, #1
        SWI 0

.data
.equ X, 23
.equ Y, 4
...
$ ./div
$ echo $?
5
$
```

Structured programming: if

```
IF exp1 op exp2 THEN
 command1
ELSE
  command2
```

not(=) ➔ NE

not(!=) ➔ EQ

not(<) ➔ GE

not(<=) ➔ GT

not(>) ➔ LE

not(>=) ➔ LT

```
          Ri = exp1
          Rj = exp2
          CMP Ri,Rj
          Bnot(op) _IFALSE
          command1
          B IEND
_IFALSE:  command2
_IEND:
```

Structured programming: if

IF exp_1 *op* exp_2 THEN
command

```
        Ri = exp1
        Rj = exp2
        CMP Ri,Rj
        Bnot(op) _IEND
        command
_IEND:
```

Example: maximum

- find largest of x, y and z

```
int x, y, z, max;
if (x>y)
 if(x>z)
  max = x
 else
  max = z
else
 if (y>z)
  max = y
 else
  max = z;
```

```
.global _start
_start:
        MOV R1, #X
        MOV R2, #Y
        MOV R3, #Z
        CMP R1, R2 @ x>y?
        BGT _tryx
        CMP R2, R3 @ y>z?
        BGT _isy
        MOV R4, R3
        B _exit
_isy:
        MOV R4, R2
        B _exit
```

Example: maximum

- find largest of x, y and z

```
int x, y, z, max;
if (x>y)
 if(x>z)
  max = x
 else
  max = z
else
 if (y>z)
  max = y
 else
  max = z;
```

```
_tryx:
        CMP R1, R3 @ x>z?
        BGT _isx
        MOV R4, R3
        B _exit
_isx:
        MOV R4, R1
_exit:
        MOV R0, R4
        MOV R7, #1
        SWI 0
.data
.equ X, 3
.equ Y, 5
.equ Z, 4
```

Not enough registers?

- need to use memory for:
  - larger data structures
  - temporary storage of partial values
- stack
  - partial results e.g. during arithmetic
  - hold state & parameters during function call
- must allocate & manage all other memory explicitly

Stack

- stack pointer == R13 == SP
- descending stack
- stack pointer
  - decremented on PUSH
  - incremented on POP

– SP starts at 0x7efff7f0 for Raspberry Pi

Stack

`PUSH` {*Rd*}

`SP = SP-4`

`(*SP) =` *Rd*

i.e.

- decrement SP by 4 bytes == 1 word down
  - to next free stack location
- store *Rd* at memory address indicated by SP

Stack

`POP` {*Rd*}

Rd `= (*SP)`

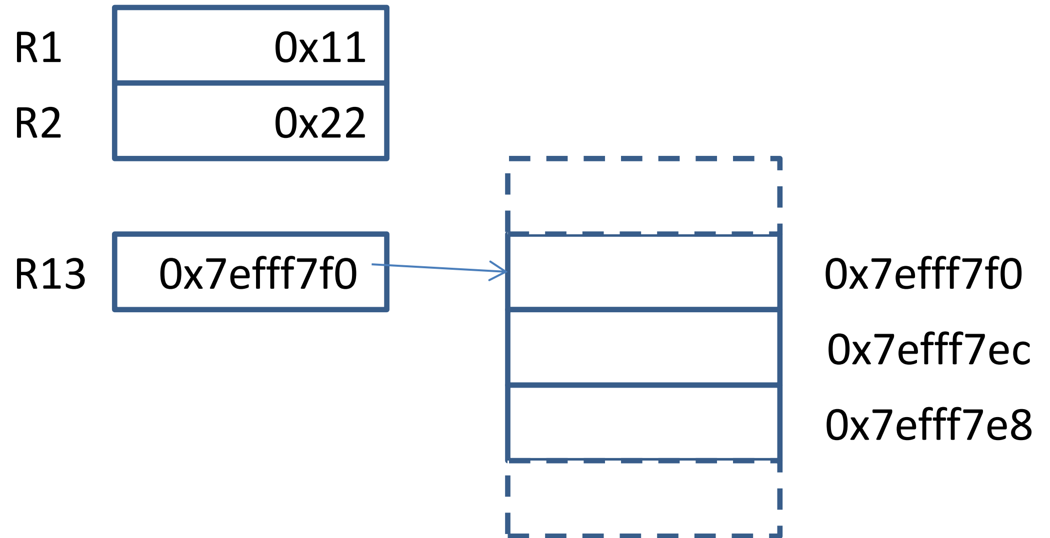
`SP = SP+4`

i.e.

- set *Rd* to contents of address indicated by SP
- increment SP by 4 bytes == 1 word up
  - next free stack location is last used

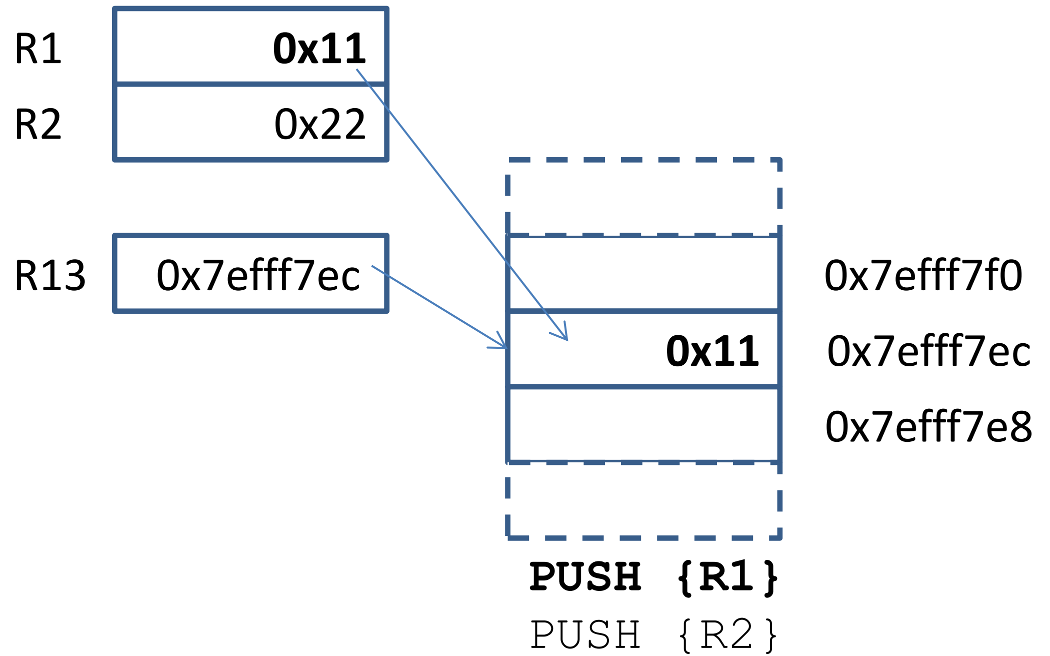
Example: swap R1 & R2

```
PUSH {R1}
PUSH {R2}
POP {R1}
POP {R2}
```



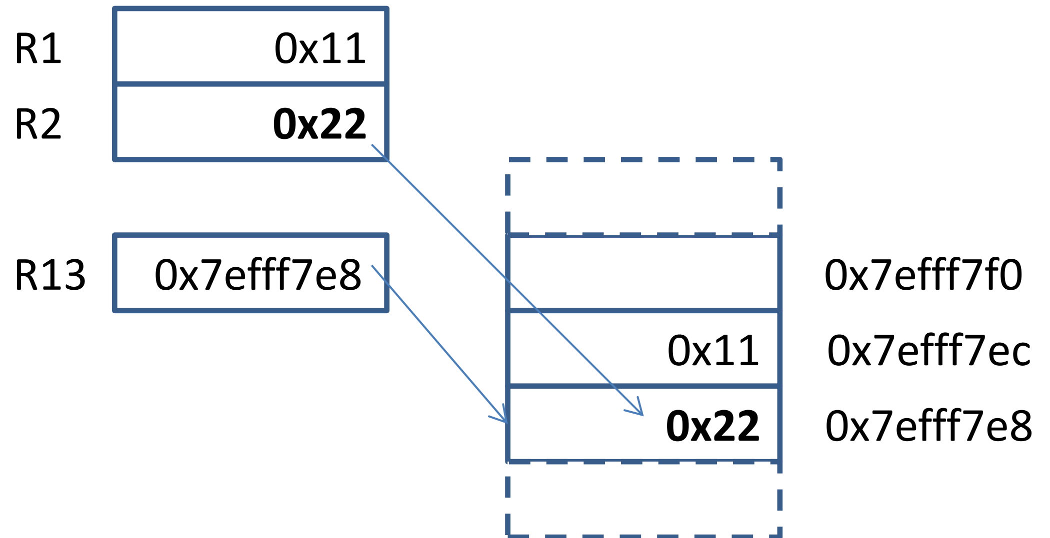
Example: swap R1 & R2

```
PUSH {R1}
PUSH {R2}
POP {R1}
POP {R2}
```



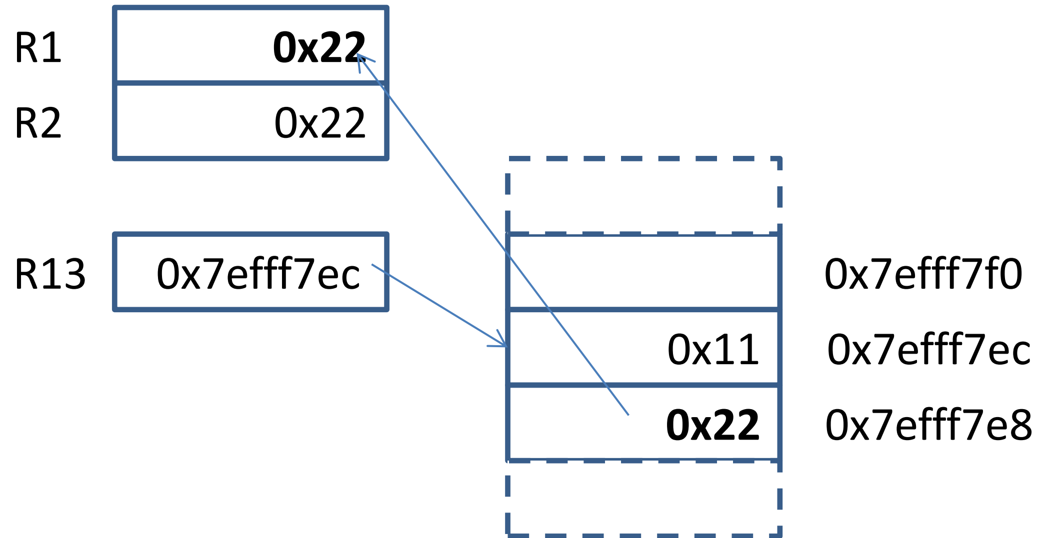
Example: swap R1 & R2

```
PUSH {R1}
PUSH {R2}
POP {R1}
POP {R2}
```



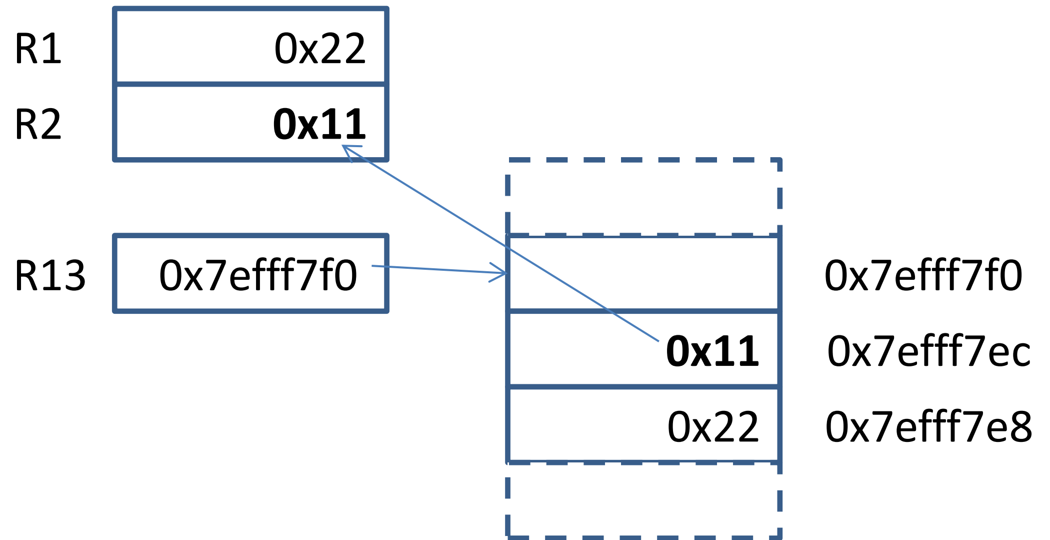
Example: swap R1 & R2

```
PUSH {R1}
PUSH {R2}
```
`POP {R1}`
```
POP {R2}
```



Example: swap R1 & R2

```
PUSH {R1}
PUSH {R2}
POP {R1}
POP {R2}
```



Evaluation order

- may wish to rearrange order of evaluation
- to optimise register use/number of instructions

$exp_1 + exp_2 \rightarrow exp_2 + exp_1$

e.g. A+B*C ➔ B*C+A

$exp_1 * exp_2 \rightarrow exp_2 * exp_1$

e.g. A*(B+C) ➔ (B+C)*A

$exp_1 * exp_2 + exp_1 * exp_3 \rightarrow exp_1 * (exp_2 + exp_3)$

e.g. X*X+X*Y ➔ X*(X+Y) ➔ (X+Y)*X

Evaluation order

- sometimes need to preserve left to right order
- e.g. expression contains function calls that change shared variables

```
int inc(int * x)
{ *x = *x+1; return *x; }
int a;
a = 2;
a+inc(&a) → 2+3 → 5
inc(&a)+a → 3+3 → 6
```

Example: (a-b)-((c-d)-((e-f)-(g-h)))

```
MOV R1,#128
MOV R2,#64
MOV R3,#32
MOV R4,#16
MOV R5,#8
MOV R6,#4
MOV R7,#2
MOV R8,#1
```

- strict left to right
- suppose only R9 & R10 spare

```
SUB R9,R1,R2
```

- R9 == A-B

```
SUB R10,R3,R4
```

- R10 == C-D
- need register for E-F
- put A-B on stack

```
PUSH {R9}
```

- *SP == A-B

```
SUB R9,R5,R6
```

- R9 == E-F
- need register for G-H
- put C-D on stack

Example: (a-b)-((c-d)-((e-f)-(g-h)))

```
PUSH {R10}
```

- *SP == C-D

```
SUB R10,R7,R8
```

- R10 == G-H

```
SUB R9,R10
```

- R9 == (E-F)-(G-H)
- R10 free
- get C-D from stack

```
POP {R10}
```

- R10 == C-D

```
SUB R10,R9
```

- R10 == (C-D)-((E-F)-(G-H))
- R9 free
- get A-B from stack

```
POP {R9}
```

- R9 == A-B

```
SUB R9,R10
```

- R9 == (A-B)-((C-D)-((E-F)-(G-H)))

Allocating memory

- after `.data`

label `.byte` *byte values* separated by ,s

- *label* is associated with address of first byte

e.g. `maxb:   .byte 0xff`

label `.word` *word values* separated by ,s

- *label* is associated with address of first byte

e.g. `maxw: .word 0xffffffff`

Displaying memory

- in `gdb`:

  `i variables`

  – display addresses for variables

  – NB start address of `.data` will change depending on how much code is before it!

  `x/`*integer*`w` *address*

  – display *integer* words from *address*

  – `x/`*integer*`b` *address*

  – display *integer* bytes from *address*

Memory access

- cannot access memory directly
- load register with memory address
- access memory indirect on register

- load register with absolute address using `MOV`

`LDR` *Rd*`,` `=`*label* ➔

- load *Rd* with address corresponding to *label*

Memory access

[*Rd*] == indirection

- use contents of *Rd* as address

LDR *Rt*, [*Rn*] ➔

Rt = **Rn*

- i.e. load *Rt* from memory whose address is in *Rn*

STR *Rt*, [*Rn*] ➔

**Rn* = *Rt*

- i.e. store *Rt* at memory whose address is in *Rn*

Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

```
...
_exit:
        MOV R0, #0
        MOV R7, #1
        SWI 0

.data
X: .word 0x16
Y: .word 0x21
T: .word 0x00
```

Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

```
   .global _start
_start:
        LDR R1, =X
        LDR R2, =Y
        LDR R3, =T
        LDR R4, [R1]
        STR R4, [R3]
        LDR R4, [R2]
        STR R4, [R1]
        LDR R4, [R3]
        STR R4, [R2]
...
```

Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

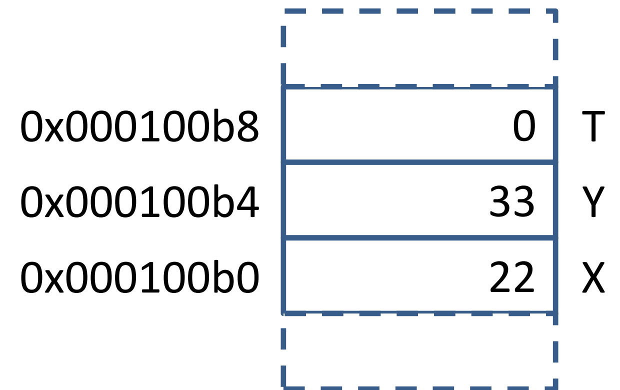
```
(gdb) i variables
All defined variables:

Non-debugging symbols:
0x000100b0  X
0x000100b4  Y
0x000100b8  T
...
(gdb) x/3w 0x100b0
0x100b0 <X>:    22        33        0
(gdb)
```

Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

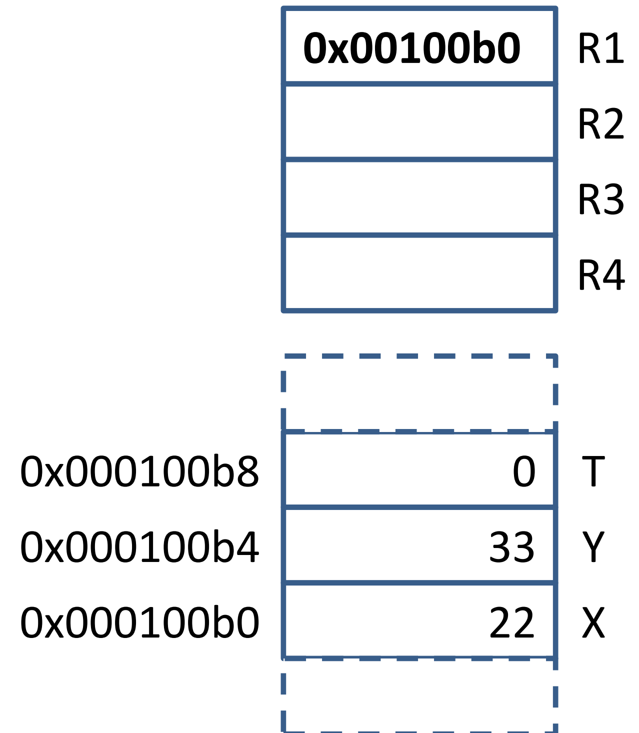
```
...
.data
X: .word 0x16
Y: .word 0x21
T: .word 0x00
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

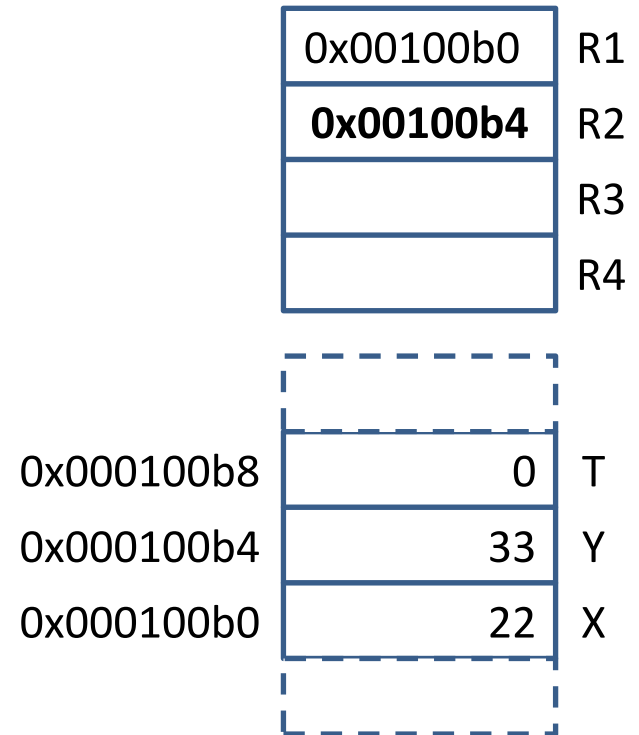
```
LDR R1, =X
LDR R2, =Y
LDR R3, =T
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

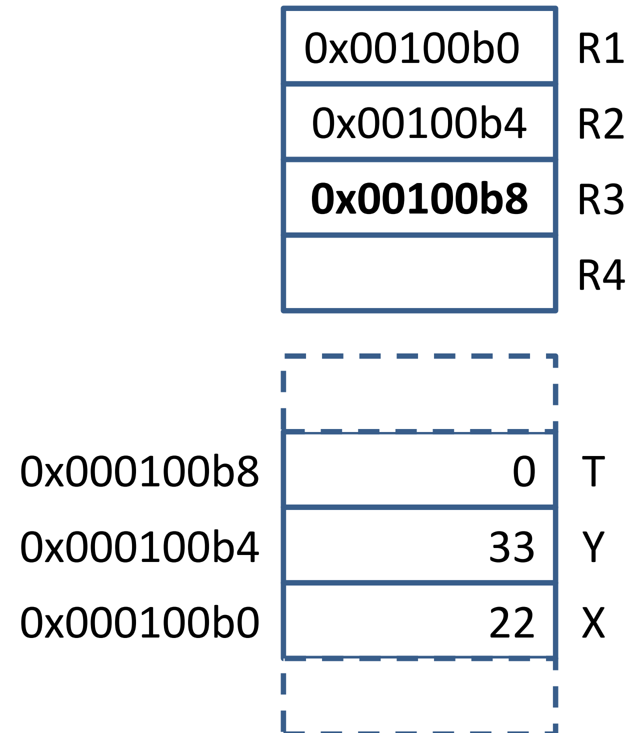
```
LDR R1, =X
LDR R2, =Y
LDR R3, =T
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

```
LDR R1, =X
LDR R2, =Y
LDR R3, =T
```

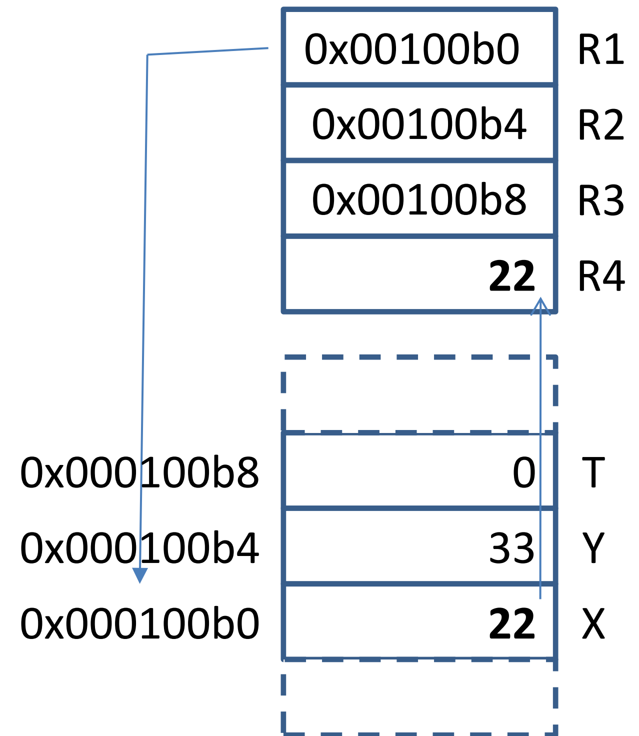


Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

```
LDR R4,[R1]
STR R4,[R3]
LDR R4,[R2]
STR R4,[R1]
LDR R4,[R3]
STR R4,[R2]
```

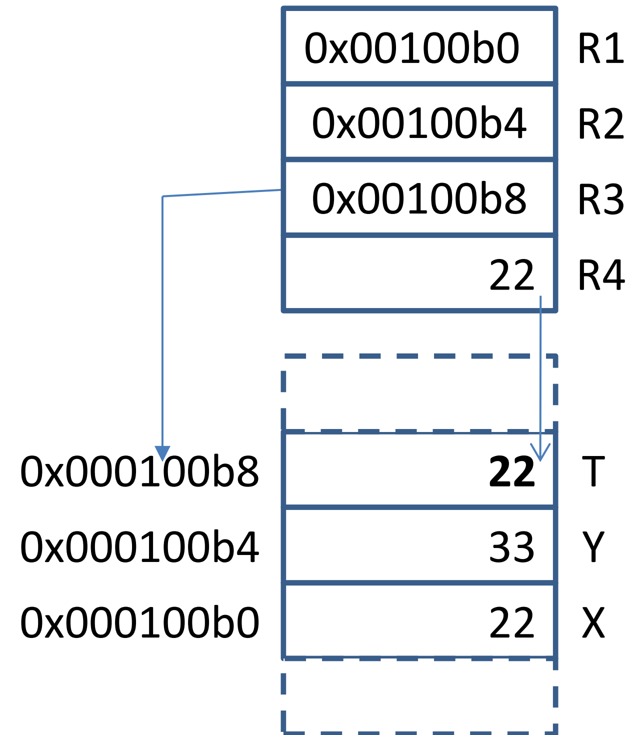
LDR R4,[R1]



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

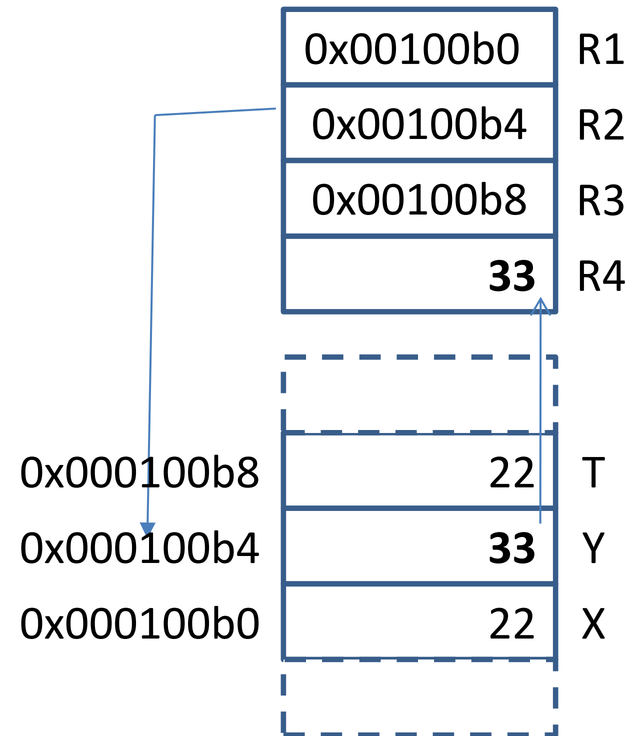
```
LDR R4,[R1]
STR R4,[R3]
LDR R4,[R2]
STR R4,[R1]
LDR R4,[R3]
STR R4,[R2]
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

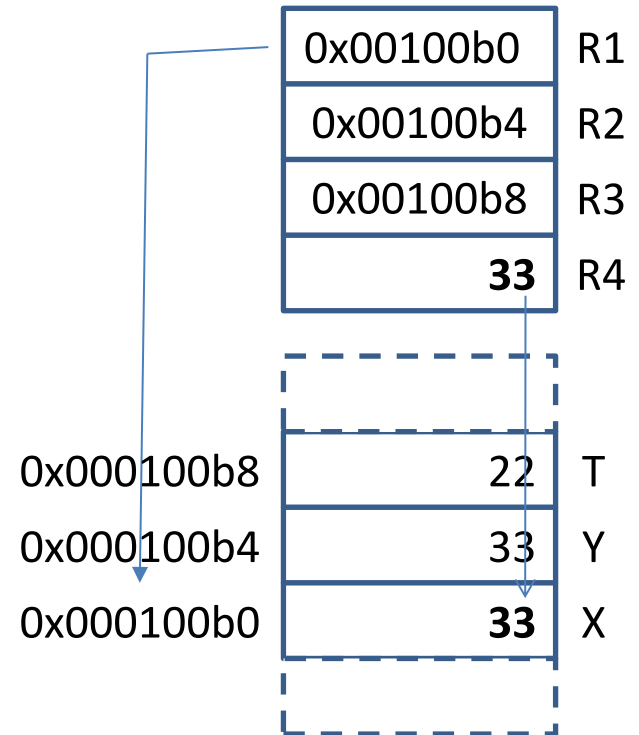
```
LDR R4,[R1]
STR R4,[R3]
LDR R4,[R2]
STR R4,[R1]
LDR R4,[R3]
STR R4,[R2]
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

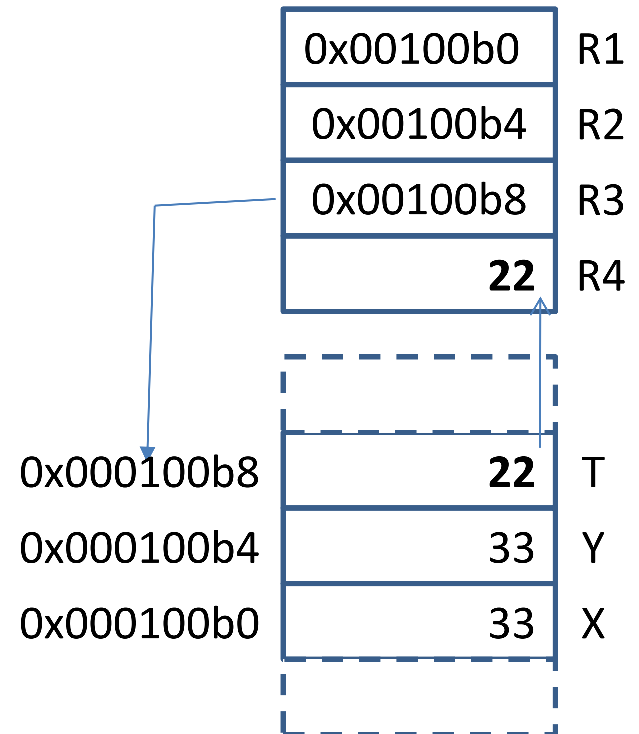
```
LDR R4,[R1]
STR R4,[R3]
LDR R4,[R2]
STR R4,[R1]
LDR R4,[R3]
STR R4,[R2]
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

```
LDR R4,[R1]
STR R4,[R3]
LDR R4,[R2]
STR R4,[R1]
LDR R4,[R3]
STR R4,[R2]
```



Example: swap a & b

```
int x;
int y;
int t;
x = 22;
y = 33;
t = x;
x = y;
y = t;
```

```
LDR R4,[R1]
STR R4,[R3]
LDR R4,[R2]
STR R4,[R1]
LDR R4,[R3]
STR R4,[R2]
```