# COL216
# Computer Architecture

## ARM Assembly Programming
## continued
## 17th Jan, 2022

# Instruction Formats in ARM

Formats: DP, DT, branch, swi

Constants:

- Range of values?
- Signed or unsigned?

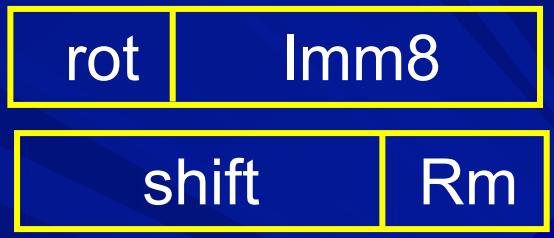
Addresses:

- Absolute or relative?
- Byte or word?

# DP instructions

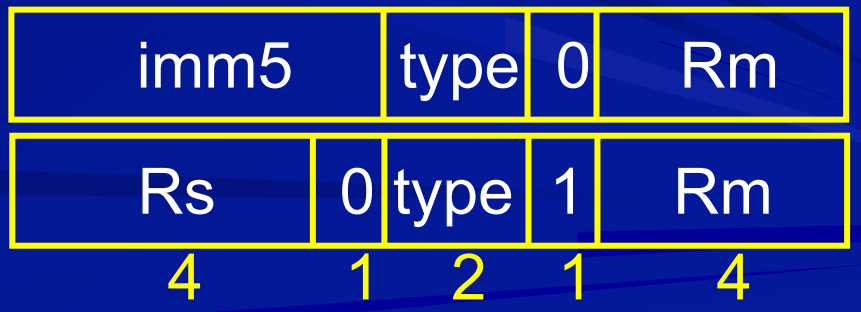

12 bit operand2

- 8 bit unsigned, 4 bit rotate
- 4 bit reg no., 8 bit shift spec




Shift amount

- 5 bit unsigned const
- 4 bit reg no. (Rs≠15)



# Multiply instructions


With or without accumulation

Result 32 or 64 bits

Signed or unsigned multiplication

R15 can’t be used

Rd ≠ Rm

# DT instructions


6 opcode bits specify I, P, U, B, W, L

12 bit offset field in DT instructions

- 12 bit unsigned constant

or

- 4 bit register number, 8 bit shift specification (same as constant shift spec of DP instructions)

# Branch instructions


L = 0 b, beq, bne ….

L = 1 bl, bleq, blne ....

24 bit immediate field in branch instructions

- signed offset
- w.r.t. PC, (address of current instruction + 8)
- multiplied by 4 and sign extended (word offset, not byte)

# Condition codes and flags

| | | |
|---|---|---|
| 0 | eq | Z = 1 |
| 1 | ne | Z = 0 |
| 2 | hs / cs (C set) | C = 1 |
| 3 | lo / cc (C clear) | C = 0 |
| 4 | mi (minus) | N = 1 |
| 5 | pl (plus) | N = 0 |
| 6 | vs (V set) | V = 1 |
| 7 | vc (V clear) | V = 0 |

| | | |
|---|---|---|
| 8 | hi | C = 1 and Z = 0 |
| 9 | ls | C = 0 or Z = 1 |
| 10 | ge | N = V |
| 11 | lt | N ≠ V |
| 12 | gt | N = V and Z = 0 |
| 13 | le | N ≠ V or Z = 1 |
| 14 | al | flags ignored |

# Condition code – general use

- Use condition codes for conditional /predicated execution of any instruction, not just branch

Example 1:

```
if (i == j)          cmp  r1,r2
                     bne  L
   h = i + j;        add  r3,r1,r2
k = k – i;        L: sub  r4,r4,r1
```

# Condition code – general use

- Use condition codes for conditional /predicated execution of any instruction, not just branch

Example 1:

```
if (i == j)          cmp  r1,r2        cmp    r1,r2
                     bne  L
  h = i + j;         add  r3,r1,r2     addeq  r3,r1,r2
k = k – i;        L: sub  r4,r4,r1     sub    r4,r4,r1
```

# Condition code – general use

Example 2:

```
if (i == j)            cmp r1,r2
                        bne L1
   h = i + j;          add r3,r1,r2
else                    b     L2
   h = i - j;      L1: sub r3,r1,r2
k = k – i;         L2: sub r4,r4,r1
```

# Condition code – general use

Example 2:

```
if (i == j)          cmp r1,r2          cmp    r1,r2
                      bne L1
   h = i + j;        add r3,r1,r2       addeq r3,r1,r2
else                  b     L2
   h = i - j;     L1: sub r3,r1,r2      subne r3,r1,r2
k = k – i;        L2: sub r4,r4,r1      sub    r4,r4,r1
```

# Which instructions modify flags?

- Compare, Test  (obviously!)
- Additionally (if you want)
  - all other DP instructions (add => adds)
  - mul and mla instructions (mul => muls)
  - Flags are affected if S = 1
- But not
  - DT instructions
  - b and bl instructions

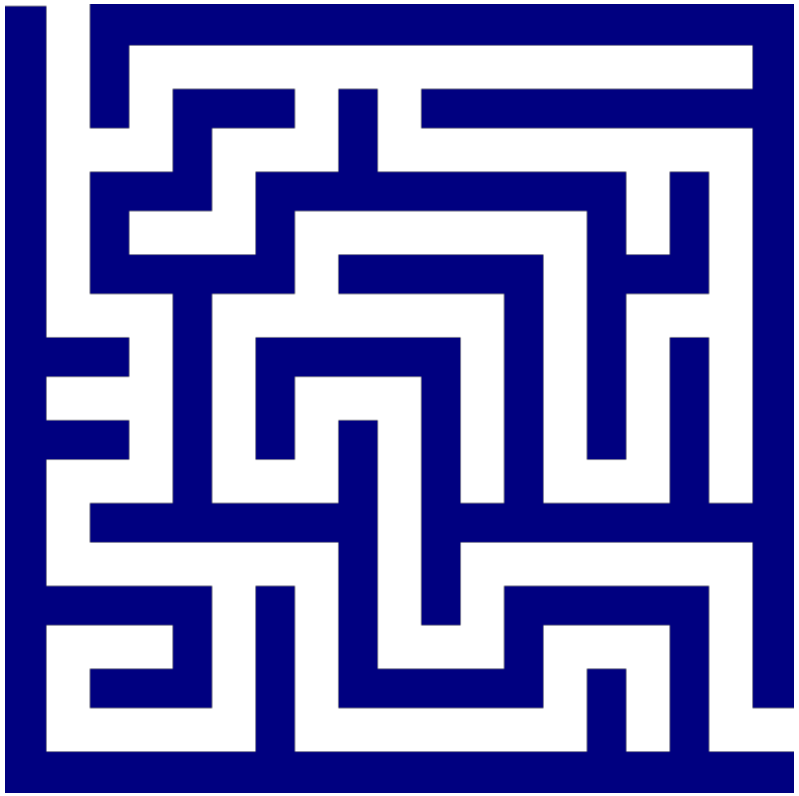
# SWI instruction


Software Interrupt

used for invoking OS function

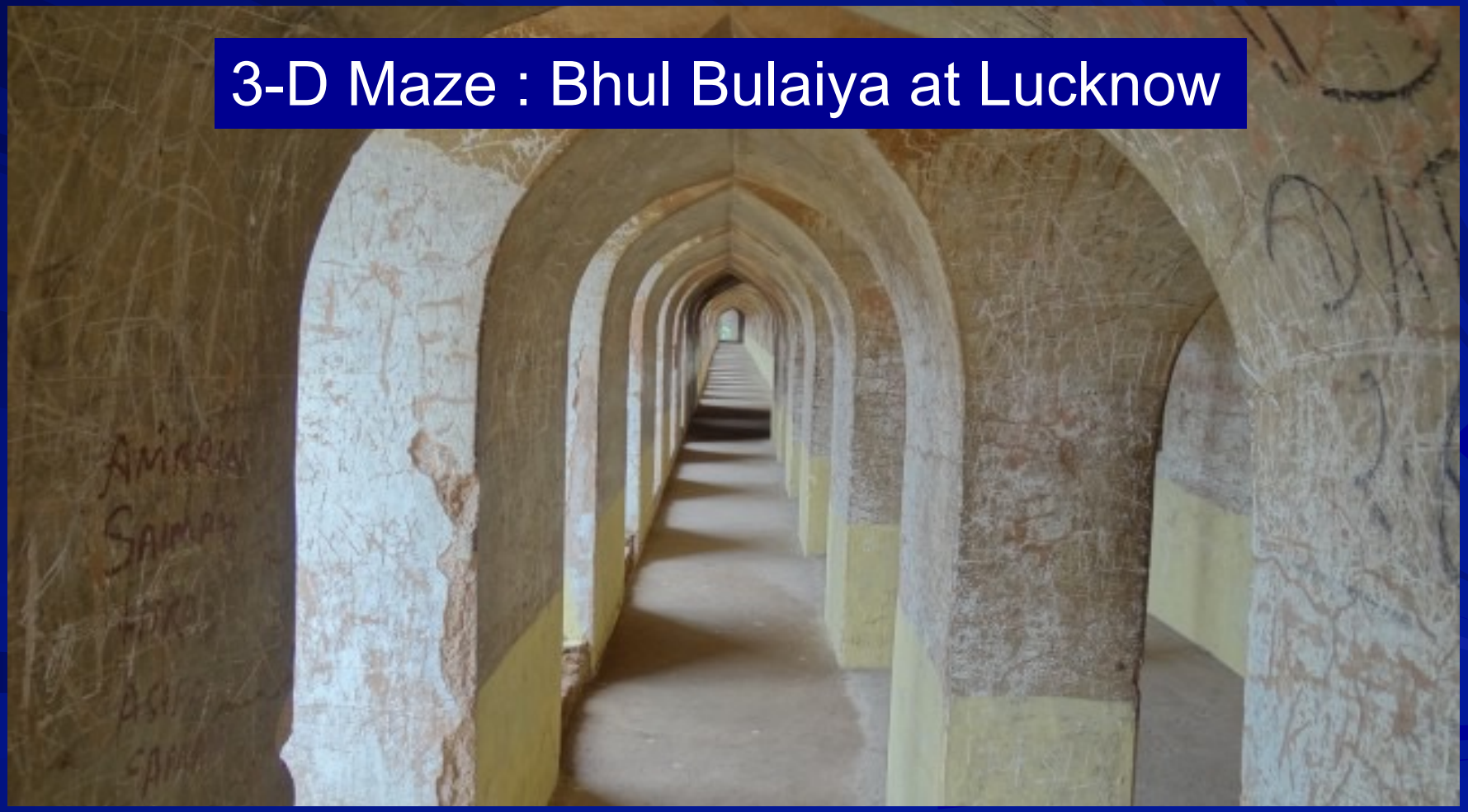
# Recursive function example

- Solving a MAZE



# Real Life Maze

# Real Life Maze

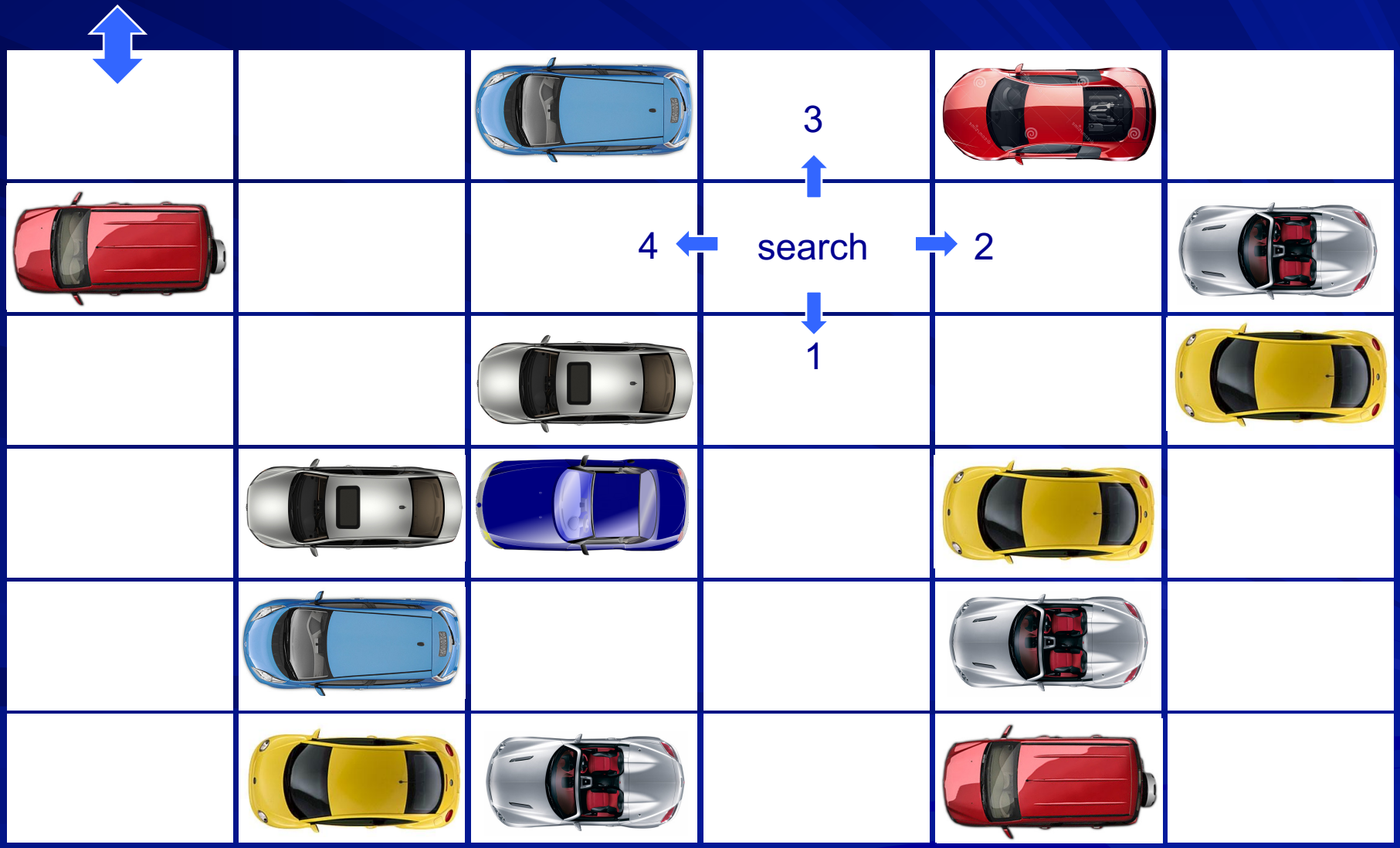


# Real Life Maze

# Maze to be actually solved by computer

# Maze to be actually solved by computer

# 6x6 Parking floor



# 6x6 Parking floor

B B B B B B B B

B S O X O X O B

B X O O O O X B

B O O X O O X B

B O X F O X O B

B O X O O X O B

B O X X O X O B

B B B B B B B B

# Recursive search

```
int search (int i, j) {
        if (Visited[i, j] == 1)  return (0);
        Visited[i, j] = 1;
        if  (Maze[i, j] == 'B'|| Maze[i, j] == 'X')  return (0);
        if  (Maze[i, j] == 'F') {print (i); print (j); return (1)};
        if  (search (i+1, j) == 1) {print ('N'); return (1)};
        if  (search (i, j+1) == 1) {print ('W'); return (1)};
        if  (search (i-1, j) == 1) {print ('S'); return (1)};
        if  (search (i, j-1) == 1) {print ('E'); return (1)};
        return (0)
};
```

# Steps

```
int search (int i, j){
        v = V[i, j];
        if (v == 1)
                return (0);
        V[i,  j] = 1;
        q = M[i, j];
        if (q == 'B')
                return (0);
        if (q == 'X')
                return (0);
        if (q == 'F'){
                print (i);
                print (j);
                return (1)};
        p = search (i+1, j);
        if (p == 1){
                print ('N');
                return (1)};
        p = search (i, j+1);
        if (p == 1){
                print ('W');
                return (1)};
        p = search (i-1, j);
        . . . .
        p = search (i, j-1);
        . . . .
        . . . .
        return(0)};
```

# Assembly program - 1

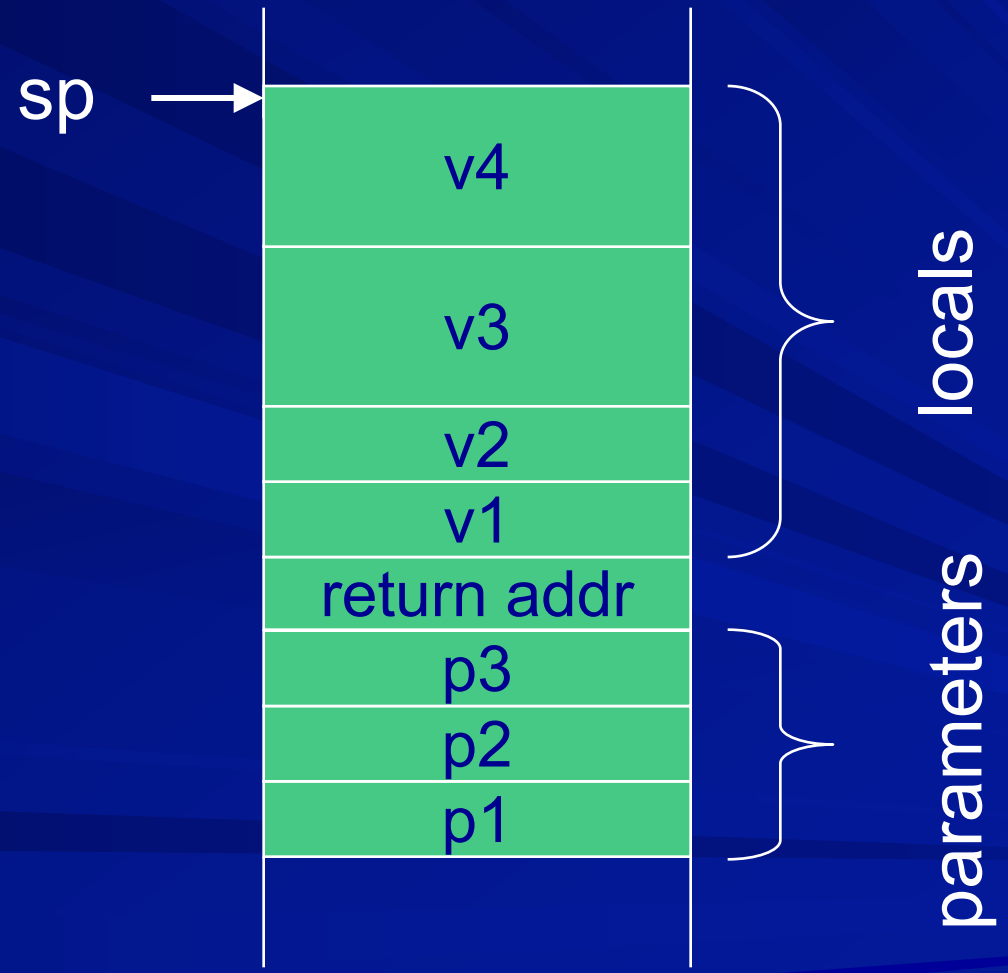
```
int search (int i, j){

        v = V[i, j];


        if (v == 1)
                return (0);
        V[i,  j] = 1;
```

```
search: str lr, [sp,#-4]!
        str r0, [sp,#-4]!
        str r1, [sp,#-4]!
        add r2, r1, r0, LSL #3
        ldr r3, =V
        ldrb r4, [r3, r2]
        cmp r4, #1
        beq Ret0
        mov r4, #1
        strb r4, [r3, r2]
```

# Activation record / stack frame



# Assembly program - 2

```
q = M[i, j];                    ldr r3, =M
                                ldrb r4, [r3, r2]
if (q == ‘B’)                   cmp r4, #’B
        return (0);             beq Ret0
if (q == ‘X’)                   cmp r4, #’X
        return (0);             beq Ret0
if (q == ‘F’){                  cmp r4, #’F
                                bne L1
        print (i);              add r0, r0, #’0
                                print r0
        print (j);              add r0, r1, #’0
                                print r0
        return (1)};            b Ret1
```

# Assembly program - 3

```
p = search(i+1,j);

if(p == 1){

        print('N');

        return(1)};
```

```
L1:     ldr r0, [sp,#4]
        ldr r1, [sp]
        add r0, r0, #1
        bl search
        cmp r0, #1
        bne L2
        mov r0, #'N
        print r0
        b Ret1
L2:
```

# Assembly program - 4

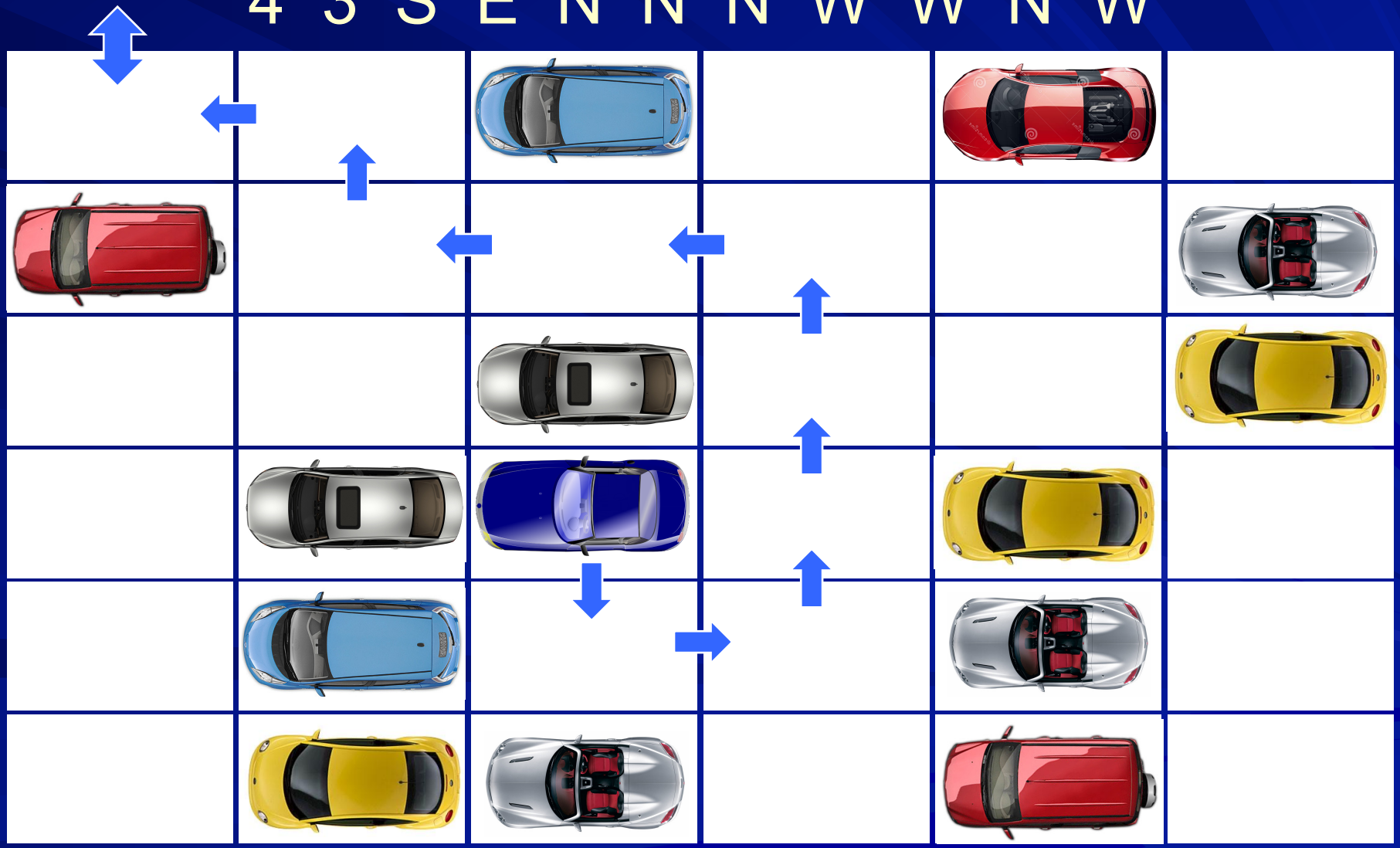
```
return (0);                 Ret0:   mov r0, #0
                                    ldr lr, [sp, #8]
                                    add sp, sp, #12
                                    mov pc, lr
return (1);                 Ret1:   mov r0, #1
                                    ldr lr, [sp, #8]
                                    add sp, sp, #12
                                    mov pc, lr
```

# The data

```
        .data
M:      .ascii "BBBBBBBBBBSOXOXOBBXOOO . . . . . . . . . ."
V:      .word 0,0,0,0,  0,0,0,0,  0,0,0,0,  0,0,0,0
        .end
```

# Result

4 3 S E N N N W W N W



# What happens in this case?

Thank you