COL216
Computer Architecture

More assembly programming

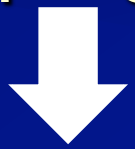
20th Jan, 2022

How to develop assembly program

- High level program

- Low level program in C

- Assembly program



- Machine language program

Low Level Program in C

- Split complex expressions into expressions involving one operation only
- Re-write for loops as simpler loops
  - while ( ) { }
  - do { } while ( ) (preferable)
- Re-write switch – case statements as multi-way branches (may use tables)
- Access arrays using pointers

Complex expression examples

```
x[ i ] = x[ i ] * z + c ;
```

```
t1 = x[ i ] ;
t2 = t1 * z ;
t2 = t2 + c ;
x[ i ] = t2 ;
```

Complex expression examples

```
x[ j ] += y[ j – i ] * z[ i ] + c ;
```

```
q = j – i ;
t1 = y[ q ] ;
t2 = z[ i ] ;
t3 = t1 * t2 ;
t1 = x[ j ] ;
t1 += t3 ;
t1 += c ;
x[ j ] = t1 ;
```

Loop examples

```
for (i = 0 ; i < max ; i++) {
    statements
} ;
```

```
i = 0 ;
while (i < max) {
    statements
    i++ ;
} ;
```

Loop examples

```
for (i = 0 ; i < max ; i++) {
    statements
} ;
```

```
i = 0 ;
do {
    statements
    i++ ;
} while (i < max) ;
```

Arrays and pointers

```
void make_zero (int * x) {


   for (i = 0 ; i < max ; i++) {
      x[ i ] = 0 ;
   } ;
};
. . . .
int A[ max ];


make_zero (A) ;
```

```
void make_zero (int * x) {
   int * r ;
   r = x + max ;
   do {
     * x++ = 0 ;
   } while (x < r) ;
};
. . . .
int A[ max ] ;
int * p ;
p = & A[ 0 ] ;
make_zero (p) ;
```

Register allocation

- Conventions
  - r0-r3 used for passing parameters
  - callee can destroy r0-r3, r12, preserves the rest
- Local allocation
  - scope of registers allocation is within a function
- Global allocation
  - scope of register allocation is across all functions

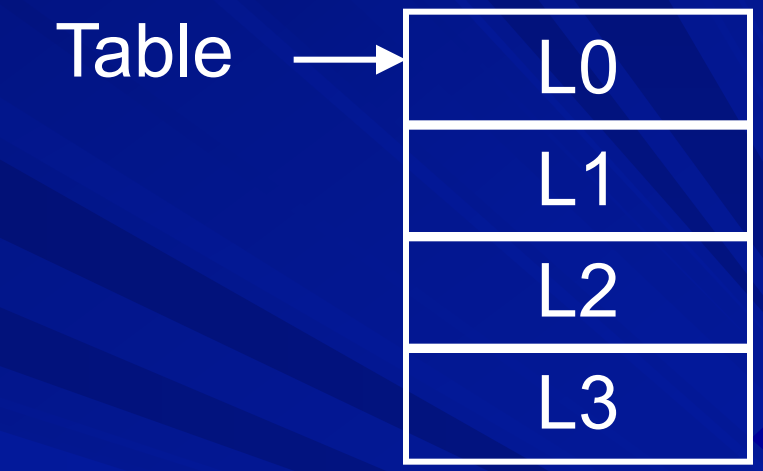
Assembly examples : switch

```
switch (k) {
    case 0:
        stmt_seq_0 ;
        break ;
    case 1:
        stmt_seq_1 ;
        break ;
    case 2:
        stmt_seq_2 ;
        break ;
    case 3:
        stmt_seq_3 ;
} ;
stmt_seq_4
```

```
            multi-way branch
back:  seq4
            . . . .
L0:
            seq0
            b back
L1:
            seq1
            b back
L2:
            seq2
            b back
L3:
            seq3
            b back
```

Table of addresses

```
        .text
        . . . .
L0:
        . . . .
L1:
        . . . .
L2:
        . . . .
L3:
        . . . .
        .data
Table:  .word L0, L1, L2, L3
```



Multi-way branch using table

```
ldr r4, =k
ldr r5, [r4]
ldr r2, =Table              @ address of table
ldr pc, [r2, r5, LSL #2]
```

Input / Output

I/O in ARMSim 1.91

Opcode	Description and Action	Inputs	Outputs	EQU
swi 0x00	Display Character on Stdout	r0: the character		SWI_PrChr
swi 0x02	Display String on Stdout	r0: address of a null terminated ASCII string	(see also 0x69 below)	
swi 0x11	Halt Execution			SWI_Exit
swi 0x12	Allocate Block of Memory on Heap	r0: block size in bytes	r0:address of block	SWI_MeAlloc
swi 0x13	Deallocate All Heap Blocks			SWI_DAlloc
swi 0x66	Open File (mode values in r1 are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0:file handle If the file does not open, a result of -1 is returned	SWI_Open
swi 0x68	Close File	r0: file handle		SWI_Close
swi 0x69	Write String to a File or to Stdout	r0: file handleor Stdout r1: address of a null terminated ASCII string		SWI_PrStr
swi 0x6a	Read String from a File	r0: file handle r1: destination address r2: max bytes to store	r0: number of bytes stored	SWI_RdStr
swi 0x6b	Write Integer to a File	r0: file handle r1: integer		SWI_PrInt
swi 0x6c	Read Integer from a File	r0: file handle	r0: the integer	SWI_RdInt
swi 0x6d	Get the current time (ticks)		r0: the number of ticks (milliseconds)	SWI_Timer

I/O in ARMSim 2.1

R0	R1[a]	Description	Operands in Memory (at address provided by R1)
0x01	M	Open a File	Filename address; filename length; file mode
0x02	M	Close a File	File handle
0x05	M	Write to File	File handle; buffer address; number of bytes to write
0x06	M	Read from File	File handle; buffer address; number of bytes to read
0x09	M	Is a TTY?	File handle
0x0A	M	File Seek	File handle; offset from file start
0x0C	M	File Length	File handle
0x0D	M	Temp File Name	Buffer address; unique integer; buffer length
0x0E	M	Remove File	Filename address; filename length
0x0F	M	Rename a File	Filename 1 address; length 1; Filename 2 address; length 2
0x10	–	Execution Time	
0x11	–	Absolute Time	
0x13	–	Get Error Num	
0x16	A	Get Heap Info	
0x18	Code	Exit Program	

UsefulFunctions.s

- prints -- print a string[1] to stdout
- fprints – print a string [1] to file
- fgets -- read one line from file / stdin
- strlen -- compute length of a string [1]
- atoi -- convert from ASCII to binary
- itoa -- convert from binary to ASCII

[1] null-terminated string

Print integer in radix 10

00000000	00000001	00000000	00001010

should print as 65546

convert to

00000110	00000101	00000101	00000100	00000110

and then to

00110110	00110101	00110101	00110100	00110110

Can you print in radix 2?

Bits and Bytes

Loading address in register

How does the following work ?

```
    ldr r4, = A
    . . .
A:  .space 100
```

Loading address in register

```
    ldr r4, = A          1040   ldr  r4, [r15, # 0x30]
    . . . .              . . . .  . . . .
    . . . .              1078   1100
    . . . .              . . . .  . . . .
A:  .space 100           1100   A[0]
                         1104   A[1]
                         . . . .  . . . .
```

Pseudo Instructions : logical

```
xnor    r1, r2, r3          eor     r1, r2, r3
                            mvn     r1, r1

nand    r1, r2, r3          and     r1, r2, r3
                            mvn     r1, r1

nor     r1, r2, r3          orr     r1, r2, r3
                            mvn     r1, r1
```

Pseudo Instructions : compare and set

```
slt     r1, r2, r3          cmp     r2, r3
                            movlt   r1, #1
                            movge   r1, #0

sge     r1, r2, r3          cmp     r2, r3
                            movge   r1, #1
                            movlt   r1, #0

sgeu    r1, r2, r3          cmp     r2, r3
                            mov     r1, #0
                            adc     r1, r1, #0
```

Pseudo Instruction : sign check

```
sign   r1, r2
```

```
mov     r1, r2, ASR #31
```

(result is 0 or -1)

```
mov     r1, r2, ASR #31
orr     r1, r1, #1
```

(result is +1 or -1)

Pseudo Instructions : absolute

```
abs   r1, r2
```

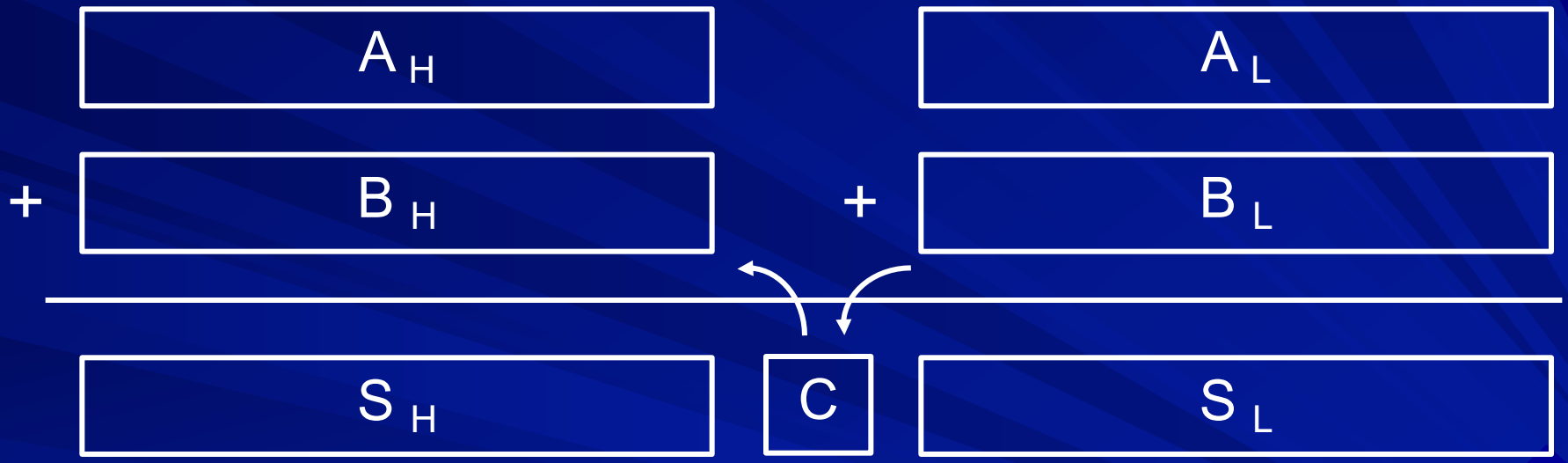
```
cmp     r2, #0
movge r1, r2
rsblt     r1, r2, #0

mov     r3, r2, ASR #31
eor       r1, r2, r3
sub       r1, r1, r3

mov     r3, r2, ASR #31
add       r1, r2, r3
eor       r1, r1, r3
```

Pseudo Instruction : swap

```
swap   r1, r2
```

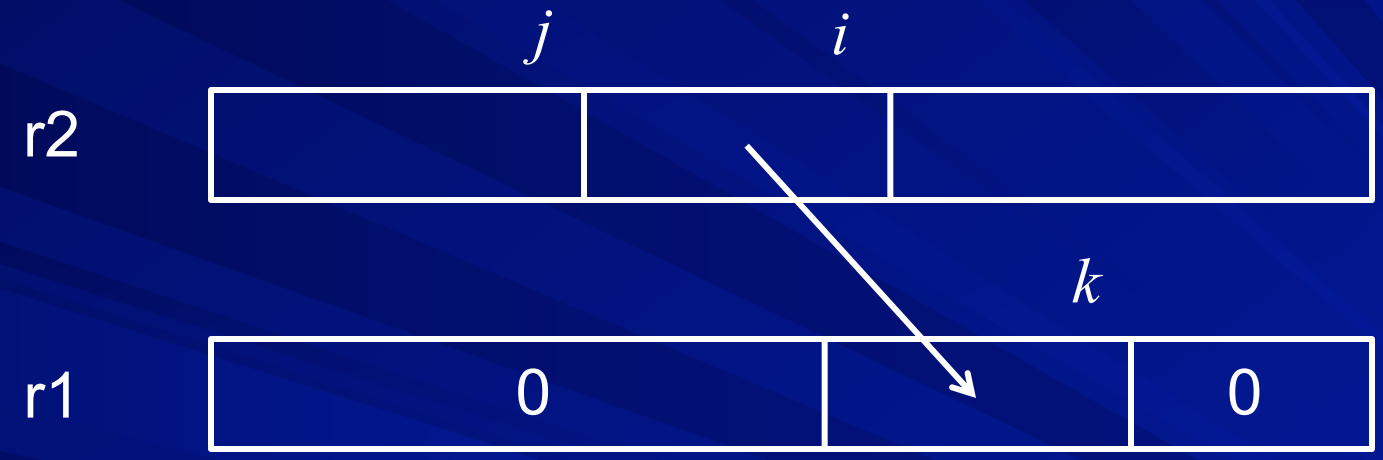
```
eor     r1, r1, r2
eor     r2, r1, r2
eor     r1, r1, r2

sub     r1, r1, r2
add     r2, r1, r2
sub     r1, r2, r1

add     r1, r1, r2
sub     r2, r1, r2
sub     r1, r1, r2
```

Double word addition



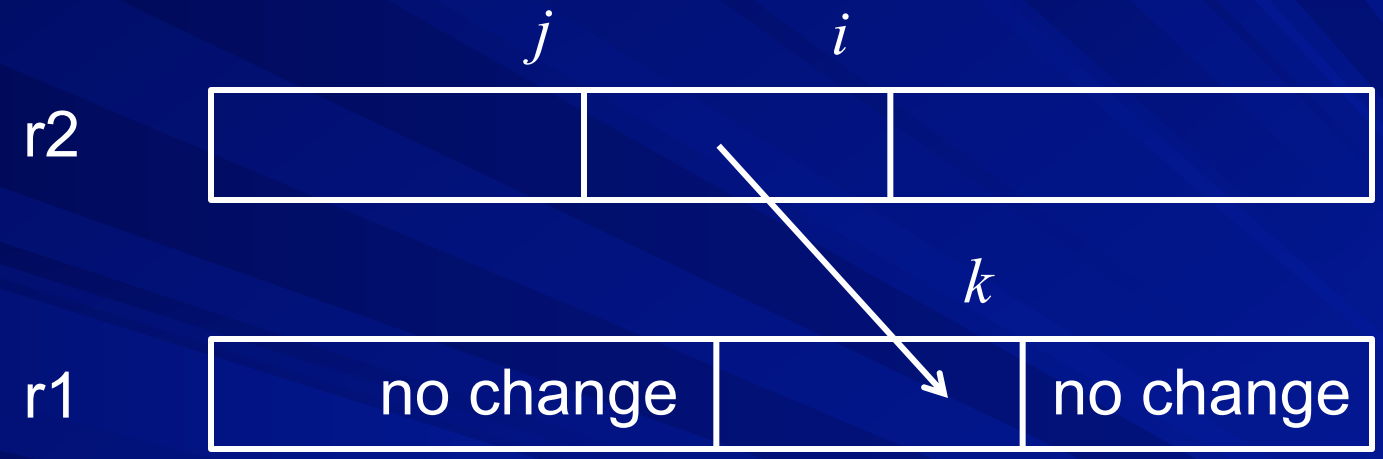

```
adds r0, r2, r4      @ A is in r3, r2
adc  r1, r3, r5      @ B is in r5, r4
                     @ S is in r1, r0
```

Field Extraction



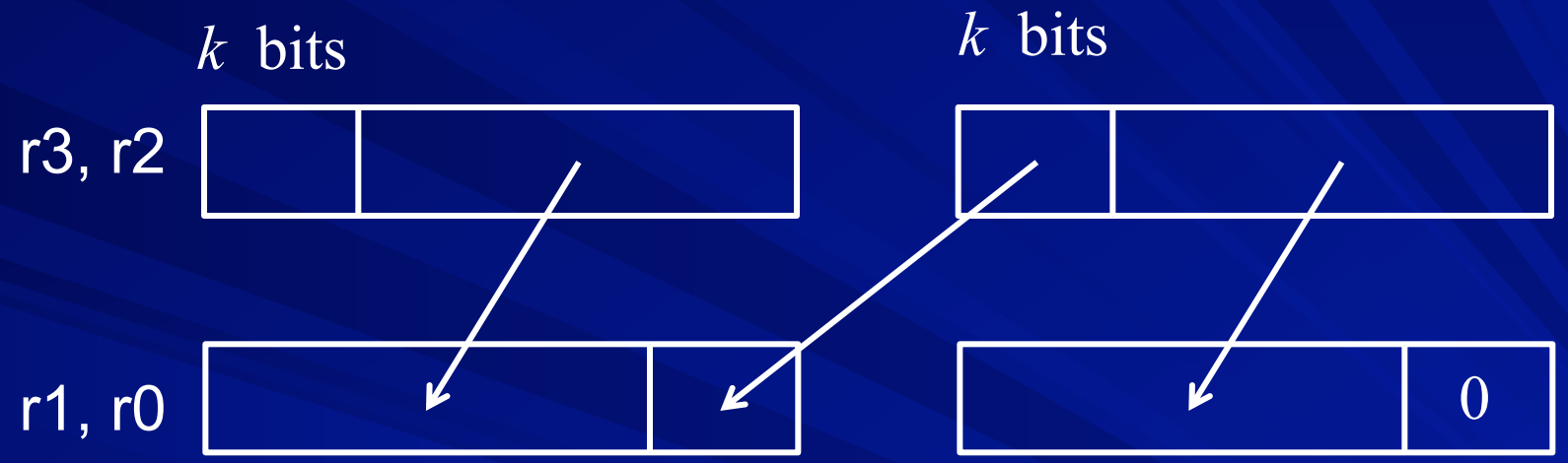

```
mov   r1, r2, LSL # 32 - j
mov   r1, r1, LSR # 32 - j + i
mov   r1, r1, LSL # k
```

Field Extraction




```
mov   r3, r2, LSL # 32 - j
mov   r3, r3, LSR # 32 - j + i
mov   r3, r3, LSL # k
mov   r1, r1, ROR # k
mov   r1, r1, LSR # j - i
mov   r1, r1, ROR # 32 - j + i - k
orr   r1, r1, r3
```

Double word shift



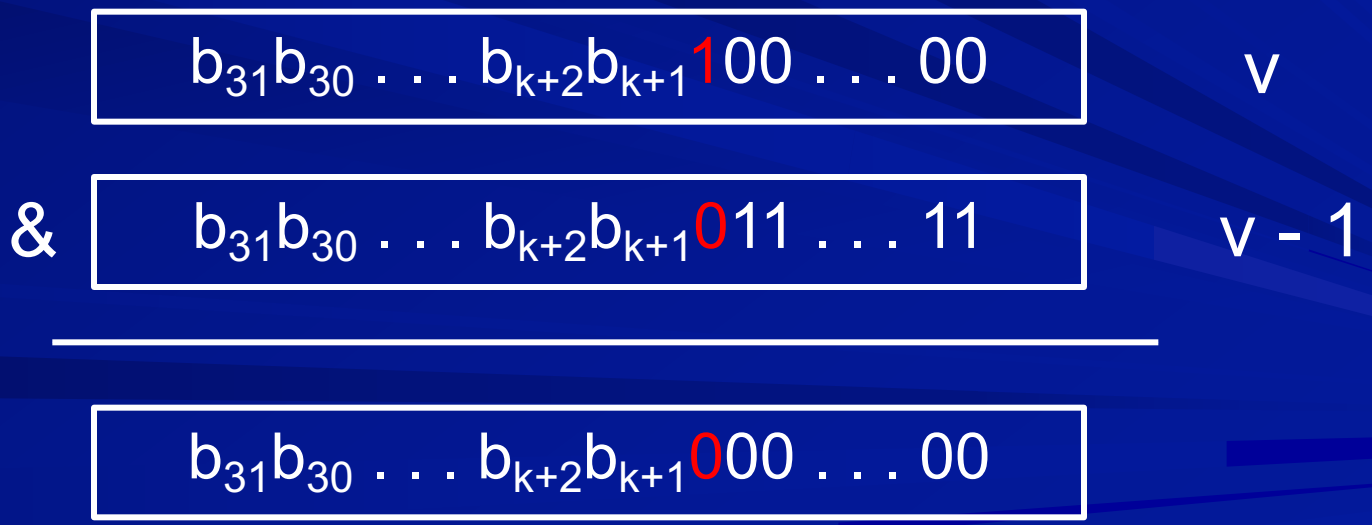

```
mov   r1, r3, LSL # k
mov   r0, r2, LSL # k
orr   r1, r1, r2, LSR # 32 - k
```

Looking at 1's in a number

- Check if there is a single 1
  - Is the number a power of 2 ?
- If number is 2^k, find k
- Count how many 1's are there
- Check if there are odd no. of 1's or even no. of 1's

Computing (v & (v - 1))

- What is special about this expression ?
- Something interesting happens to the rightmost 1 in v.



If v is a power of 2

$v = 2^k$

$\Rightarrow b_{31} = b_{30} \ldots = b_{k+2} = b_{k+1} = 0$

$\Rightarrow (v \,\&\, (v - 1)) = 0$

$v = 0$

$\Rightarrow (v \,\&\, (v - 1)) = 0$

therefore, $v \neq 0$ and $(v \,\&\, (v - 1)) = 0$

$\Rightarrow v = 2^k$ for some k

Check if power of 2

```
        movs  r0, r1          @ v is in r1
        beq   out             @ v = 0
        sub   r2, r1, #1      @ r2 = v - 1
        ands  r2, r1, r2      @ r2 = v & (v - 1)
        movne r0, #0          @ not a power of 2
        moveq r0, #1          @ power of 2
out:                          @ result in r0
```

Counting 1's

```
c = 0
while (v > 0) do {
    v &= v - 1          => clears least significant 1
    c++
}
```

Counting 1's

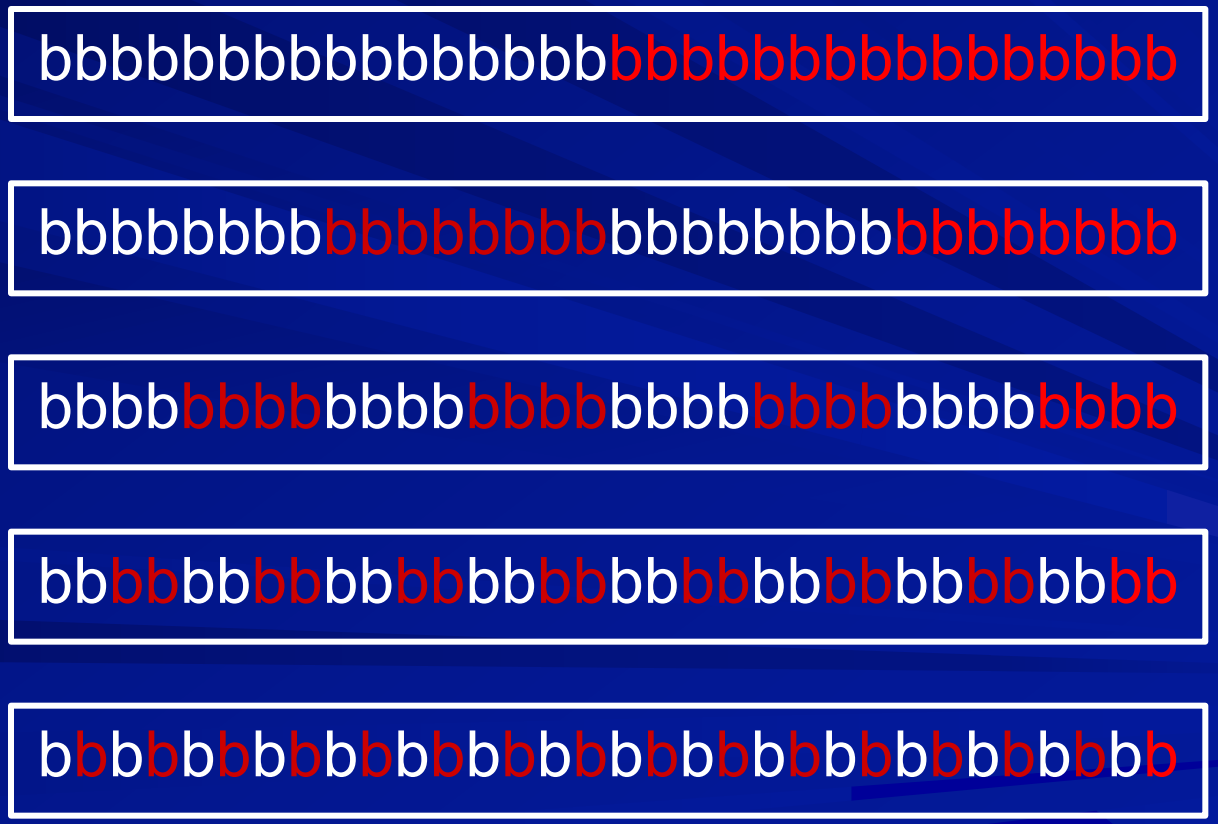
```
c = 0
while (v > 0) do {

    t = v - 1
    v &= t
    c++
}
```

```
@ c: r0,   v: r1,   t:r2
        mov  r0, #0
loop: movs r2, r1
        beq   out
        sub   r2, r2, #1
        and   r1, r1, r2
        add   r0, r0, #1
        b     loop
out:
```

Find k where v = 2^k

v has a single 1. Find its position.

Perform binary search



Find k where v = 2^k

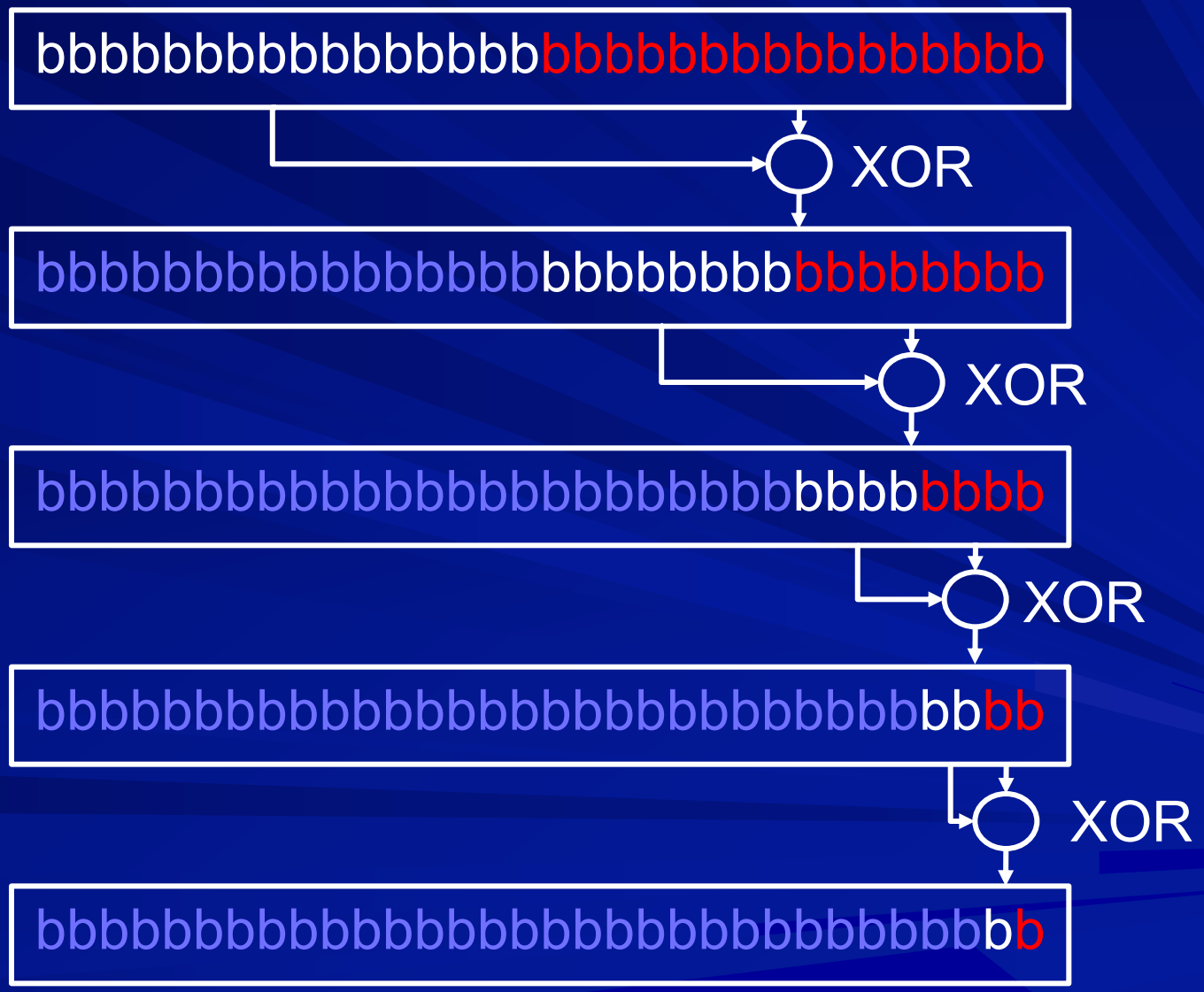
```
c = 0 ;
if (v & 0x0000FFFF = 0)
    {c += 16; v >>= 16;};
if (v & 0x000000FF = 0)
    {c += 8; v >>= 8;};
if (v & 0x0000000F = 0)
    {c += 4; v >>= 4;};
if (v & 0x00000003 = 0)
    {c += 2; v >>= 2;};
if (v & 0x00000001 = 0)
    {c += 1; v >>= 1;};
```

Find k where v = 2^k

```
c = 0 ;
if (v & 0x0000FFFF = 0)
    {c += 16; v >>= 16;};
if (v & 0x000000FF = 0)
    {c += 8; v >>= 8;};
if (v & 0x0000000F = 0)
    {c += 4; v >>= 4;};
if (v & 0x00000003 = 0)
    {c += 2; v >>= 2;};
if (v & 0x00000001 = 0)
    {c += 1; v >>= 1;};
```

```
ands   r2, r1, 0xFF
addeq r0, r0, #8
moveq r1, r1, LSR #8
```

Computing Parity



4 of the
5 steps
are shown

Computing Parity

v = v xor (v >> 16)	eor r0, r0, r0, LSR #16
v = v xor (v >> 8)	eor r0, r0, r0, LSR #8
v = v xor (v >> 4)	eor r0, r0, r0, LSR #4
v = v xor (v >> 2)	eor r0, r0, r0, LSR #2
v = v xor (v >> 1)	eor r0, r0, r0, LSR #1
v = v & 1	and r0, r0, #1

Bit reversal

- interchange two 16 bit fields
- interchange four 8 bit fields
- interchange eight 4 bit fields
- interchange sixteen 2 bit fields
- interchange thirty two 1 bit fields

Bit reversal

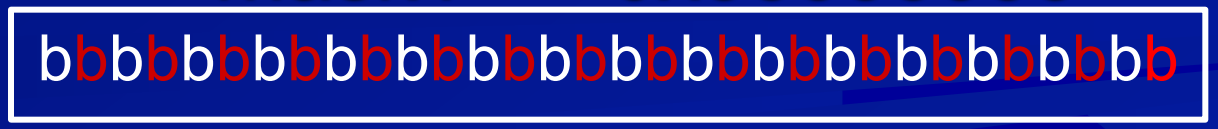
mask16 = 0x0000FFFF

mask8 = 0x00FF00FF

mask4 = 0x0F0F0F0F

mask2 = 0x33333333

mask1 = 0x55555555



Bit reversal

R = v & mask16; L = (v >> 16) & mask16
v = R << 16 | L
R = v & mask8; L = (v >> 8) & mask8
v = R << 8 | L
R = v & mask4; L = (v >> 4) & mask4
v = R << 4 | L
R = v & mask2; L = (v >> 2) & mask2
v = R << 2 | L
R = v & mask1; L = (v >> 1) & mask1
v = R << 1 | L

Bit reversal (one step)

```
                                @ v : r0,  R : r1,  L : r0,
                                @ masks : r4,r5,r6,r7,r8
R = v & mask8                   and r1, r0, r5
L = (v >> 8) & mask8            and r0, r5, r0, LSR #8
v = R << 8 | L                  orr   r0, r0, r1, LSL #8
```

Reference

- http://graphics.stanford.edu/~seander/bithacks.html

Thank you