EIE/ENE 334 Microprocessors

Lecture 9:

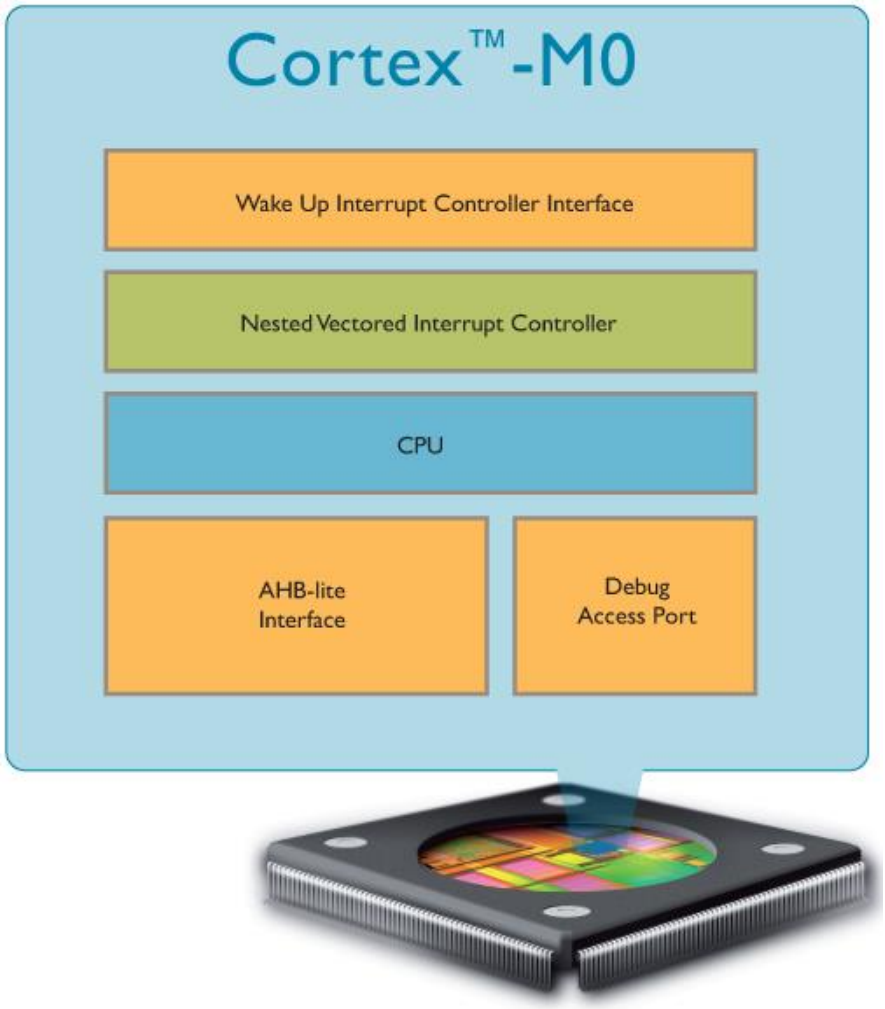
The Cortex-M0 Instruction Set

Week #09 : Dejwoot KHAWPARISUTH

Adapted from

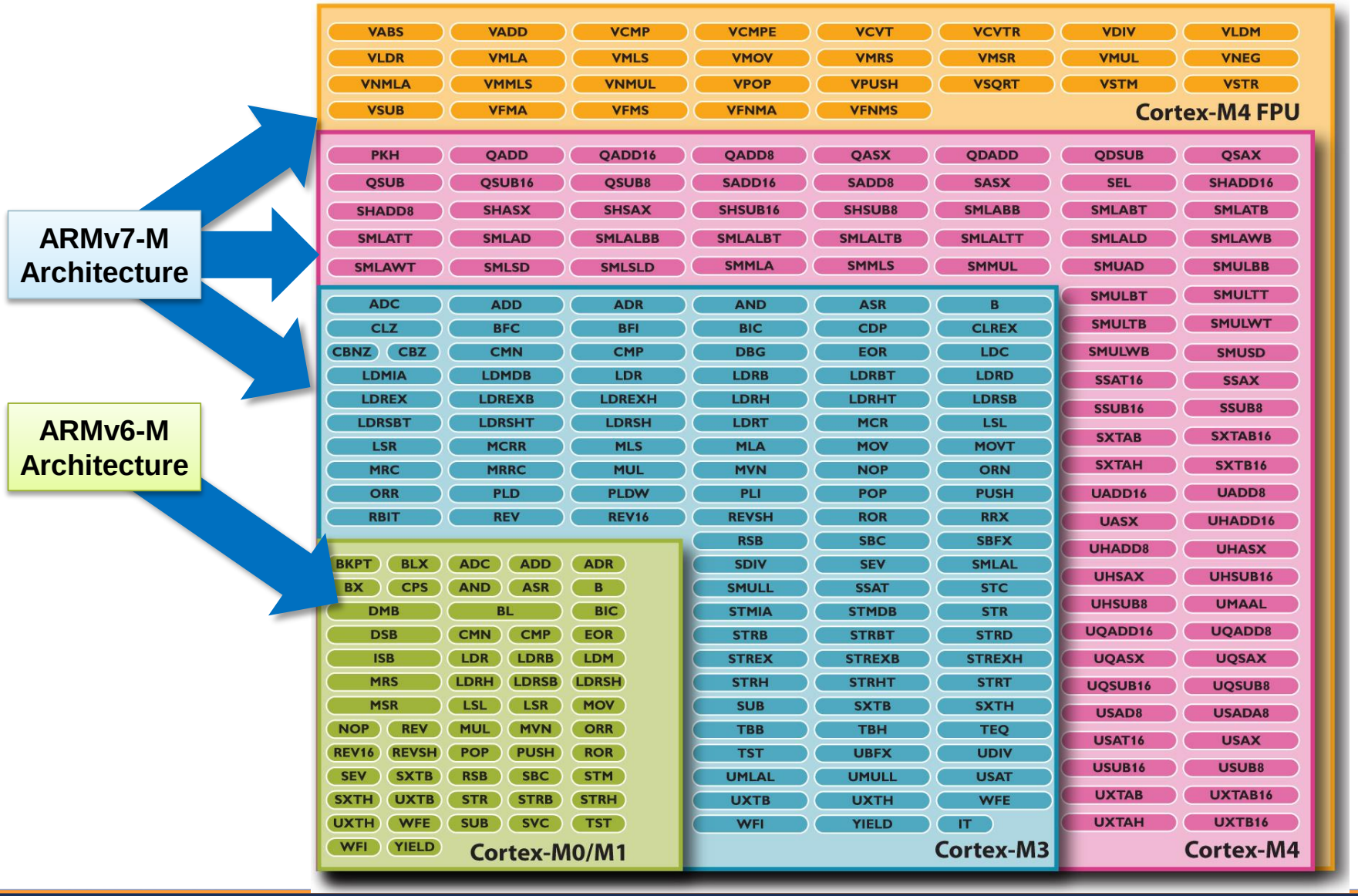
http://webstaff.kmutt.ac.th/~dejwoot.kha/

Cortex-M0



- **ARMv6-M Architecture**
  - 16-bit Thumb-2 with system control instructions
- **Fully programmable in C**
- **3-stage pipeline**
- **von Neuman architecture**
- **AHB-Lite bus interface**
- **Fixed memory map**
- **1-32 interrupts**
  - Configurable priority levels
  - Non-Maskable Interrupt support
- **Low power support**
- **Core configured with or without debug**
  - Variable number of watchpoints and breakpoints

Binary Upwards Compatibility



Instruction set

- only the 16-bit Thumb instructions
- and a minimum subset of 32-bit Thumb instructions

Assembly syntax (ARM assembler)

```
label   mnemonic   operand1, operand2,…  ; Comment
```

label -> used as a reference to an address location (optional) and data address (start at the first column in the line)

Assembly syntax (ARM assembler)

Immediate data: prefixed with "#"

example:

```
label    mnemonic   operand1, operand2,…  ; Comment

         MOVS  R0,#0x1F    ; Set R0 = 0x1F
         MOVS  R0,#'A'     ; Set R0 = 0x41 (ASCII code)
```

Assembly syntax (ARM assembler)

Constant definition:

example:

```
label    mnemonic    operand1, operand2,…   ; Comment

CLK_BA_base    EQU 0x50000200    ; 32-bits
PWRCON         EQU 0x00          ; 8-bits?
```

Assembly syntax (ARM assembler)

Embedded data:

example:

```
label    mnemonic   operand1, operand2,…  ; Comment

         LDR   R0,=MyData ; Get the address of MyData
               ; LDR here is a pseudo instruction
               ; -> a PC relative load
         LDR   R1,[R0]    ; R1 = 0x12345678

         LDR   R0,=MyText ; R0 = the starting addr
         LDR   R1,[R0]    ; R1 = 0x61434241
MyData   DCD   0x12345678
MyText   DCB   "ABCabc0123\n",0 ; Null terminated
                                ; string
```

Assembly syntax (ARM assembler)

```
    26:          LDR           R0,=MyData     ; Get the address of MyData
0x0000016E 4807      LDR       r0,[pc,#28]   ;
    27:          LDR           R1,[R0]        ; R1 = 0x12345678
    28:
0x00000170 6801      LDR       r1,[r0,#0x00]
    29:          LDR      R0,=MyText          ; R0 = the starting addr
0x00000172 4807      LDR       r0,[pc,#28]   ; @0x00000190
    30:          LDR R1,[R0]
    31:
    32:          ALIGN 4
    33: MyData   DCD        0x12345678
    34: MyText   DCB        "ABCabc0123\n",0        ; Null terminated string
    35: }
0x00000174 6801      LDR       r1,[r0,#0x00]

0x00000176 0000      MOVS      r0,r0
0x00000178 5678      LDRSB     r0,[r7,r1]
0x0000017A 1234      ASRS      r4,r6,#8
0x0000017C 4241      RSBS      r1,r0,#0
0x0000017E 6143      STR       r3,[r0,#0x14]
0x00000180 6362      STR       r2,[r4,#0x34]
0x00000182 3130      ADDS      r1,r1,#0x30
0x00000184 3332      ADDS      r3,r3,#0x32
0x00000186 000A      MOVS      r2,r1
```

Instruction set: Memory access

ADR: Generates a PC-relative address.

example:

```
    20:          ADR R0, MyData
; write address 0x0000016C to R0
; R0 = 0x0000016C
    21:
    22:          ALIGN 4
    23: MyData   DCD        0x12345678

0x00000168 A000      ADR       r0,{pc}+4
                                    ; @0x0000016C
0x0000016A 0000      MOVS      r0,r0
0x0000016C 5678      LDRSB     r0,[r7,r1]
0x0000016E 1234      ASRS      r4,r6,#8
```

Instruction set: Memory access

LDR and STR, immediate offset

example:

```
LDR R0, [R5]
; Loads R0 from the address in R5.

STR R1, [R6,#const-struc]
; const-struc is an expression evaluating
; to a constant in the range 0-1020.
```

Instruction set: Memory access

LDR and STR, register offset

example:

```
STR R0, [R5, R1]
; Store value of R0 into an address equal to
; sum of R5 and R1

LDRSH R1, [R2, R3]
; Load a halfword from the memory address
; specified by (R2 + R3), sign extend to 32-bits
; and write to R1.
```

Instruction set: Memory access

LDR, PC-relative

example:

```
    20:          LDR R0, MyData
; Load R0 with a word of data from an address MyData
; R0 = 0x12345678
    21:
    22:          ALIGN 4
    23: MyData   DCD        0x12345678

0x00000168 4800      LDR      r0,[pc,#0]
                                  ; @0x0000016C
0x0000016A 0000      MOVS     r0,r0
0x0000016C 5678      LDRSB    r0,[r7,r1]
0x0000016E 1234      ASRS     r4,r6,#8
```

Instruction set: Memory access

LDM and STM: Load and Store Multiple registers.

example:

```
LDM R0!,{R0,R3,R4}
; LDMIA,LDMFD is a synonym for LDM
; IA-Increment After
; FD-Full Descending stacks
; R0=memory[R0], R3=memory[R0+4}, R4=memory[R0+8]
; R0 = R0 + 12

STMIA R1!,{R2-R4,R6}
; memory[R1]=R2, memory[R1+4]=R3, memory[R1+8]=R4,
; memory[R1+12]=R6, and update R1
```

Instruction set: Memory access

PUSH and POP:

example:

```
PUSH {R0,R4-R7}
; Push R0,R4,R5,R6,R7 onto the stack
; PUSH -> decrement SP and store
PUSH {R2,LR}
; Push R2 and the link-register onto the stack

POP {R0,R6,PC}
; Pop r0,r6 and PC from the stack, then branch to
; the new PC.
; POP -> read and then increment
```

Instruction set: data processing

ADC, ADD, RSB, SBC, and SUB: Add with carry, Add, Reverse Subtract, Subtract with carry, and Subtract.

example: shows two instructions that add a 64-bit integer contained in R0 and R1 to another 64-bit integer contained in R2 and R3, and place the result in R0 and R1.

[R1:R0] = [R1:R0]+[R3:R2]

```
ADDS R0, R0, R2
; add the least significant words

ADCS R1, R1, R3
; add the most significant words with carry
```

Instruction set: data processing

ADC, ADD, RSB, SBC, and SUB: Add with carry, Add, Reverse Subtract, Subtract with carry, and Subtract.

example: shows the RSBS instruction used to perform a 1's complement of a single register.

```
RSBS R7, R7, #0
; subtract R7 from zero
```

S-Suffic indicate an instruction that update APSR(flags: N, Z, C, and V)

Instruction set: data processing

AND, ORR, EOR, and BIC: Logical AND, OR, Exclusive OR, and Bit Clear.

example:

```
ANDS R2, R2, R1
ORRS R2, R2, R5
ANDS R5, R5, R8
EORS R7, R7, R6
BICS R0, R0, R1
```

Instruction set: data processing

ASR, LSL, LSR, and ROR: Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, and Rotate Right.

example:

```
ASRS R7, R5, #9
; Arithmetic shift right by 9 bits
LSLS R1, R2, #3
; Logical shift left by 3 bits with flag update
LSRS R4, R5, #6
; Logical shift right by 6 bits
RORS R4, R4, R6
; Rotate right by the value in
; the bottom byte of R6.
```

Instruction set: data processing

Instruction set: data processing

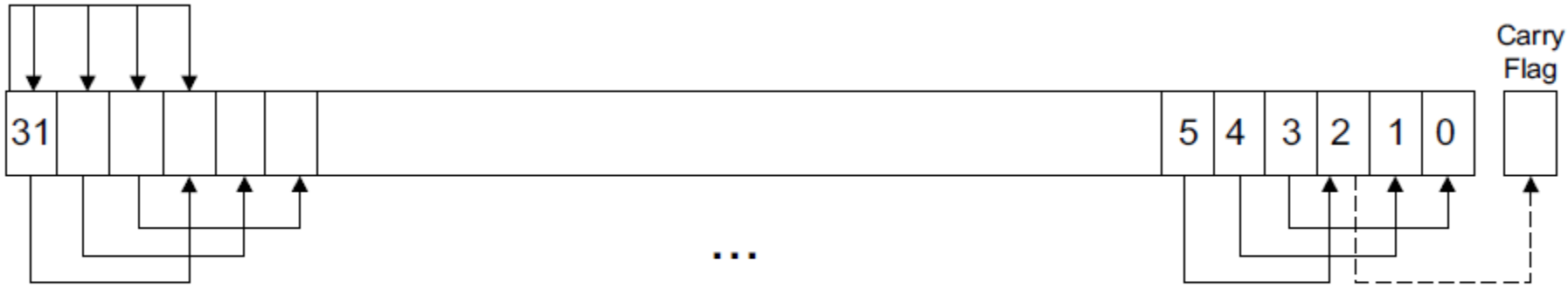


Figure 3-1 ASR #3

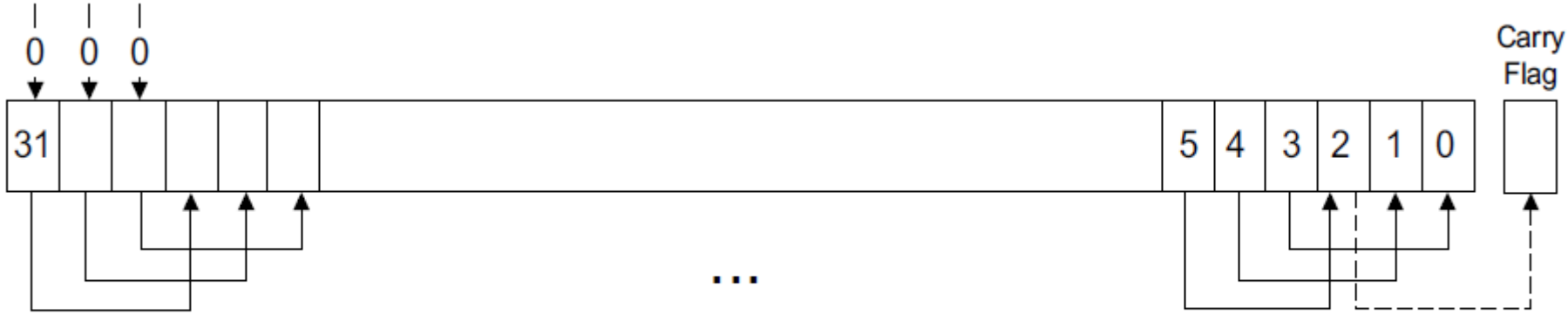


Figure 3-2 LSR #3

Instruction set: data processing

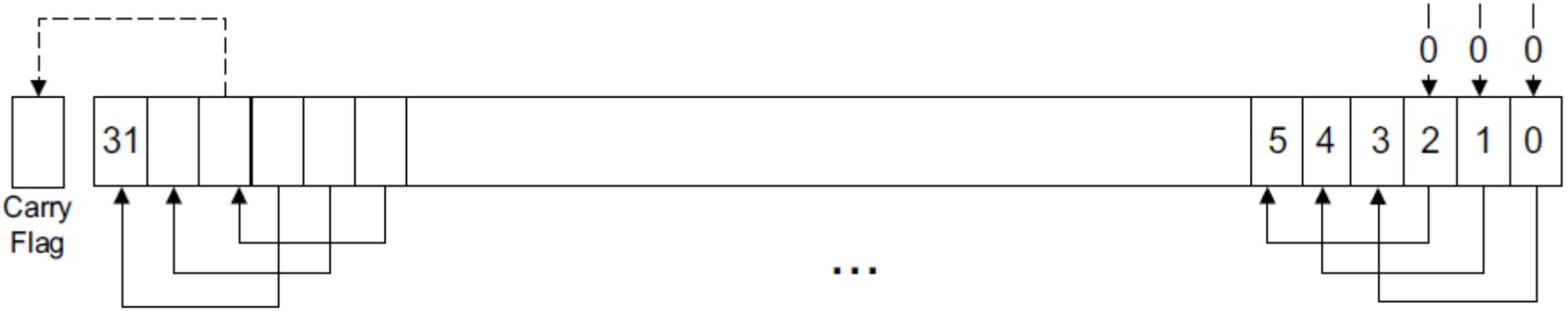


Figure 3-3 LSL #3

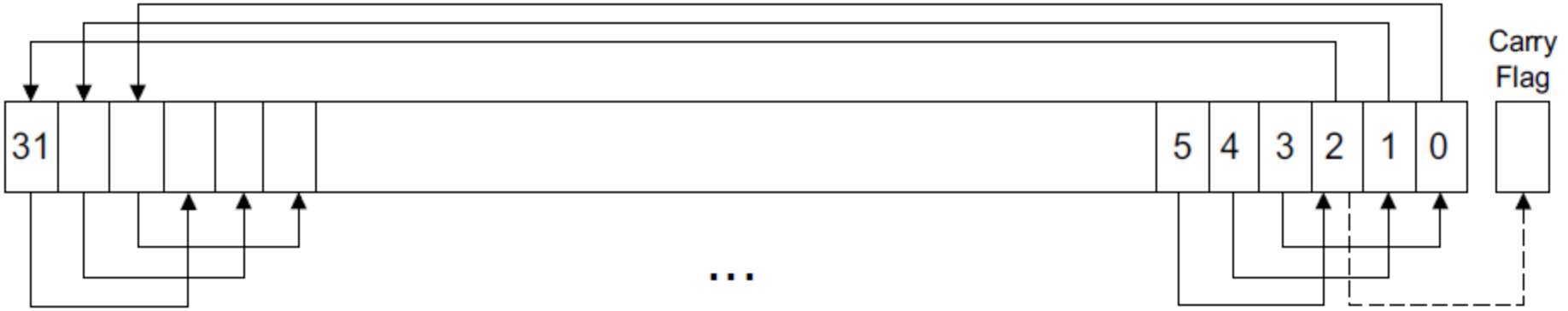


Figure 3-4 ROR #3

Instruction set: data processing

CMP and CMN: Compare and Compare Negative.

example:

```
CMP R2, R9
CMN R0, R2
```

Instruction set: data processing

MOV and MVN: Move and Move NOT.

example:

```
MOVS R0, #0x000B
; Write value of 0x000B to R0, flags get updated
MOVS R1, #0x0
; Write value of zero to R1, flags are updated
MOV R10, R12
; Write value in R12 to R10, flags are not updated
MOVS R3, #23
; Write value of 23 to R3
MOV R8, SP
; Write value of stack pointer to R8
MVNS R2, R0
; Write inverse of R0 to the R2 and update flags
```

Instruction set: data processing

MULS: Multiply using 32-bit operands, and producing a 32-bit result.

example:

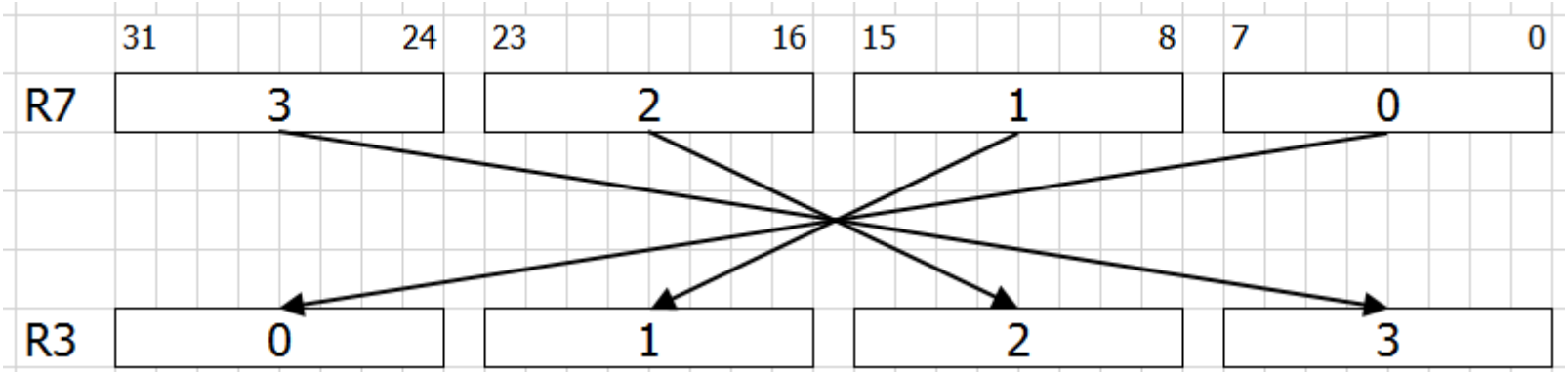
```
MULS R0, R2, R0
; Multiply with flag update, R0 = R0 x R2
```

Instruction set: data processing

REV, REV16, and REVSH: Reverse bytes.

example:

```
REV R3, R7
; Reverse byte order of value in R7 and write it
; to R3
```

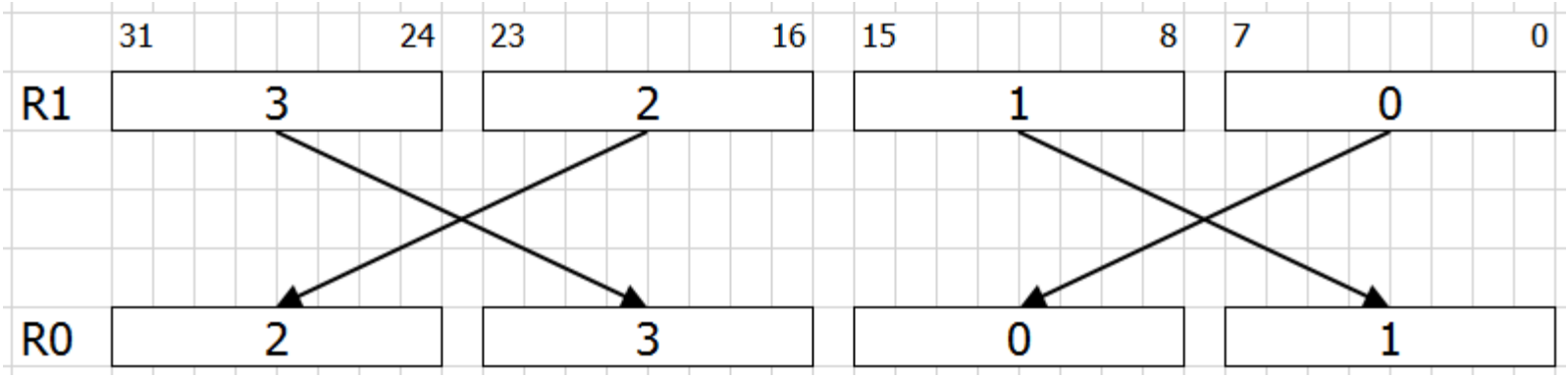


Instruction set: data processing

REV, REV16, and REVSH: Reverse bytes.

example:

```
REV16 R0, R1
; Reverse byte order of each 16-bit halfword in R0
```

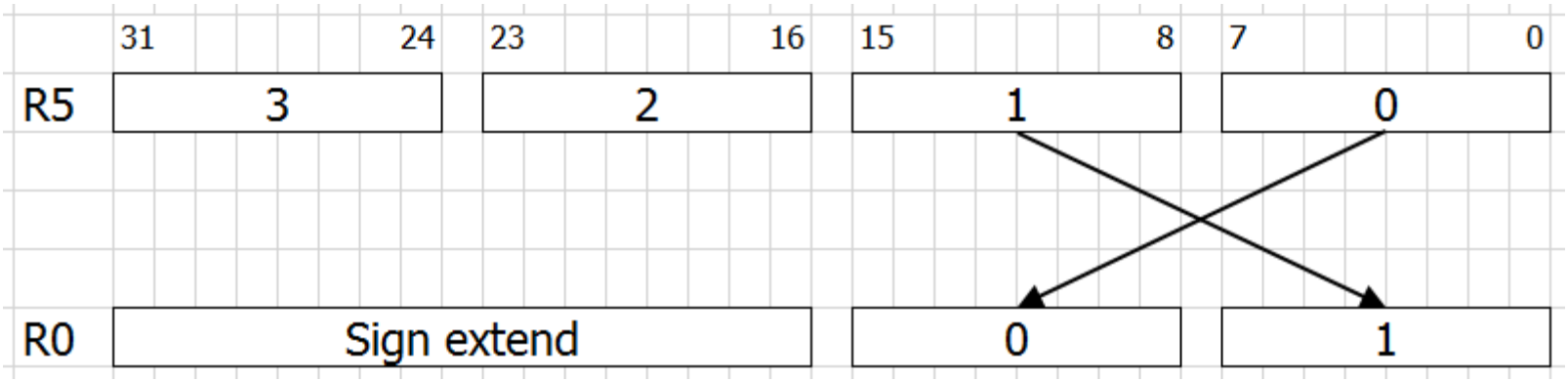


Instruction set: data processing

REV, REV16, and REVSH: Reverse bytes.

example:

```
REVSH R0, R5
; Reverse signed halfword
```



Instruction set: data processing

SXT and UXT: Sign extend and Zero extend.

example:

```
SXTH R4, R6
; Obtain the lower halfword of the
; value in R6 and then sign extend to
; 32 bits and write the result to R4.

UXTB R3, R10
; Extract lowest byte of the value in R10 and zero
; extend it, and write the result to R3
```

Instruction set: data processing

TST: Test bits.

example:

```
TST R0, R1
; Perform bitwise AND of R0 value and R1 value,
; condition code flags are updated but result is
; discarded
```

Instruction set: Branch and control

B, BL, BX, and BLX: Branch instructions.

example:

```
B loopA
; Branch to loopA
BL funC
; Branch with link (Call) to function funC, return
; address stored in LR
BX LR
; Return from function call
BLX R0
; Branch with link and exchange (Call) to a
; address stored in R0
BEQ labelD
; Conditionally branch to labelD if last flag
; setting instruction set the Z flag, else do not branch.
```

Condition code suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal, last flag setting result was zero
NE	Z = 0	Not equal, last flag setting result was non-zero
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned
LS	C = 0 or Z = 1	Lower or same, unsigned
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z = 1 and N != V	Less than or equal, signed
AL	Can have any value	Always. This is the default when no suffix is specified.

Instruction set: Miscellaneous

BKPT: Breakpoint.

; provide a breakpoint function during debug

example:

```
BKPT #0
; Breakpoint with immediate value set to 0x0.
```

Instruction set: Miscellaneous

CPS: Change Processor State.

example:

```
CPSID i
; Disable all interrupts except NMI (set PRIMASK)

CPSIE i
; Enable interrupts (clear PRIMASK)
```

Instruction set: Miscellaneous

DMB: Data Memory Barrier.

; when the memory system is complex, prevent race conditions

example:

```
DMB
; Data Memory Barrier
```

Instruction set: Miscellaneous

DSB: Data Synchronization Barrier.

example:

```
DSB
; Data Synchronization Barrier
```

Instruction set: Miscellaneous

ISB: Instruction Synchronization Barrier.

example:

```
ISB
; Instruction Synchronization Barrier
```

Instruction set: Miscellaneous

MRS: Move the contents of a special register to a general-purpose register.

example:

```
MRS R0, PRIMASK
; Read PRIMASK value and write it to R0
```

Instruction set: Miscellaneous

MSR: Move the contents of a general-purpose register into the specified special register.

example:

```
MSR CONTROL, R1
; Read R1 value and write it to the CONTROL
; register
```

Instruction set: Miscellaneous

NOP: No operation.

example:

```
NOP
; No operation
```

Instruction set: Miscellaneous

SEV: Send Event.

; normally used in multiprocessor system, to wake up (WFE)

example:

```
SEV
; Send Event
```

Instruction set: Miscellaneous

SVC: Supervisor Call.

; cause the SVC exception

example:

```
SVC #0x32
; Supervisor Call (SVC handler can extract the
; immediate value by locating it using the
; stacked PC)
```

Instruction set: Miscellaneous

WFE: Wait For Event.

```
; to enter sleep mode
```

example:

```
   WFE   ; Wait For Event
```

Instruction set: Miscellaneous

WFI: Wait for Interrupt.

; to enter sleep mode

example:

```
WFI   ; Wait for Interrupt
```