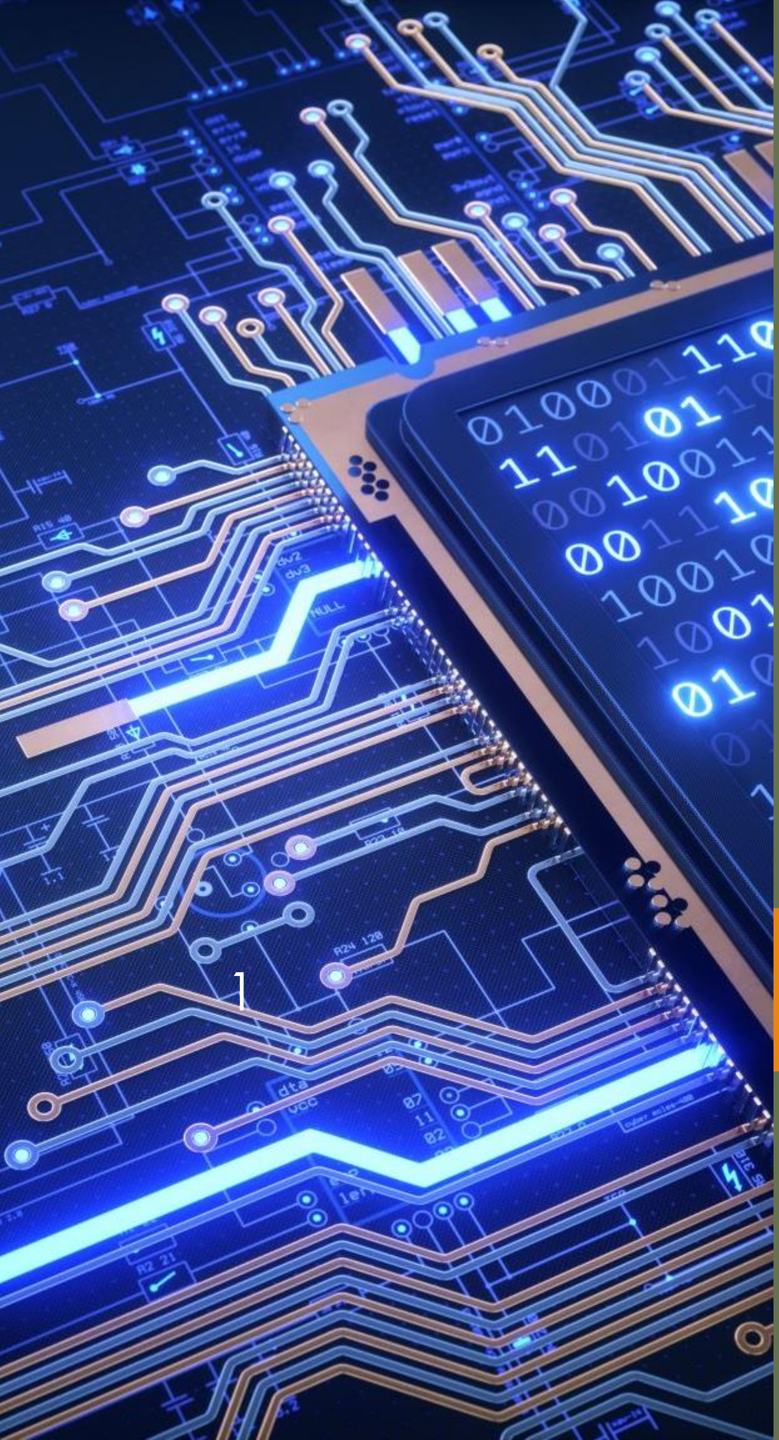


Embedded Controller programming for Real Time Systems: ECE-40097

Lesson 3

Vijay Kumar

Main Topics

- Instruction Sets
- Assembly Instructions
- Instruction Format
- Arithmetic and Logical Instructions
- Conditional Logic

Instruction Sets

- Supports large number of ARM 32 bits instructions
- Supports Thumb instruction set
  - 16 bits and 32 bits
  - Thumb provides better code density at expense of performance
  - Thumb-2 achieves high performance and better code density
  - Most 16-bit instructions can only access eight of the general-purpose registers, R0-R7 (low registers)

ARM and Thumb

ARM	Thumb-1
• 32-bit instruction set, called the ARM instructions • Powerful and good performance • Larger program memory compared to 8-bit and 16-bit processors • Larger power consumption	• 16-bit instruction set, first used in ARM7TDMI processor • Provides a subset of the ARM instructions • Better code density compared to 32-bit RISC architecture • Code size is reduced by ~30%, but performance is also reduced by ~20%

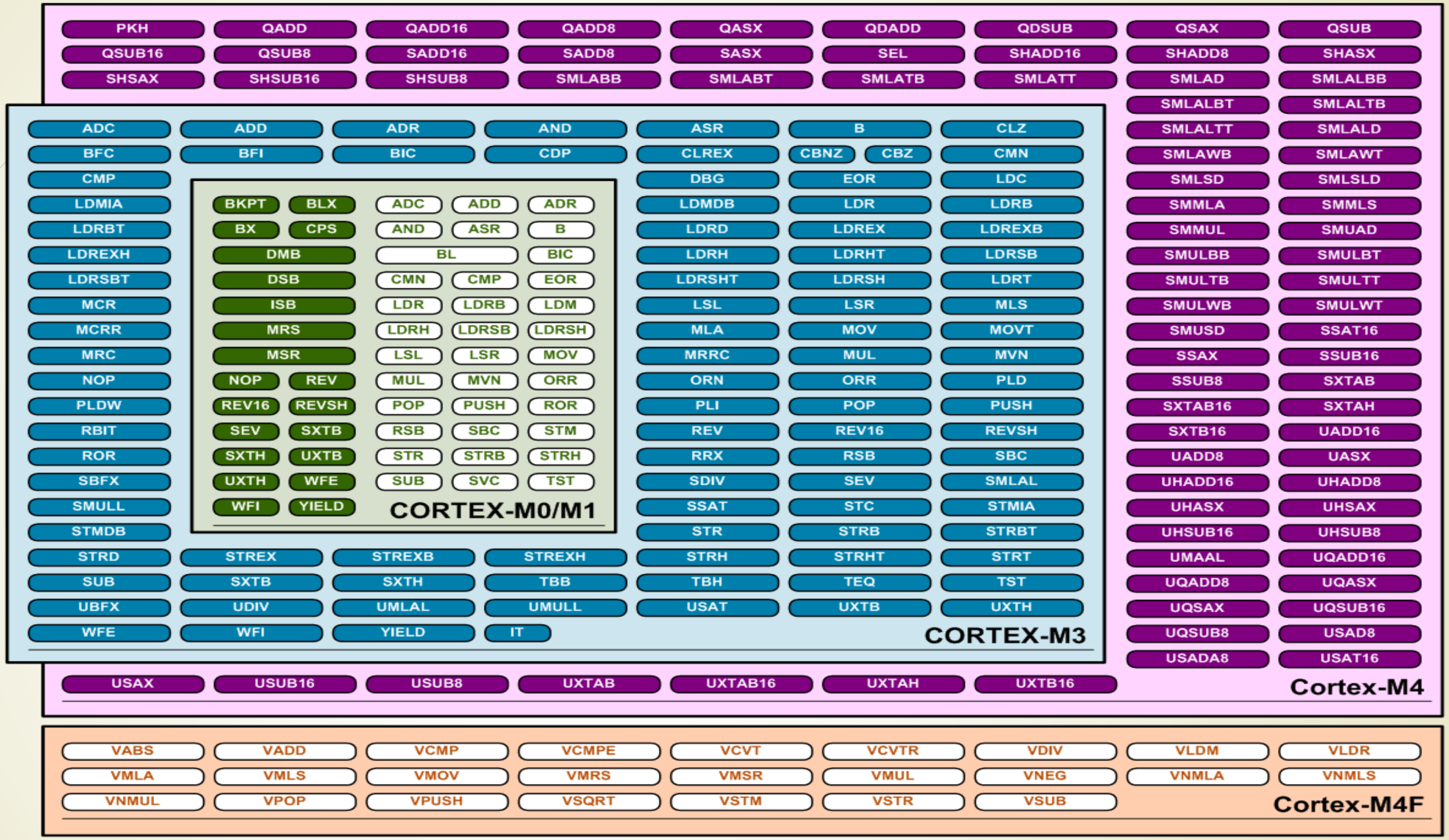
Thumb -2

- Thumb-2
  - Consists of both 32-bit Thumb and 16-bit Thumb-1 instruction sets
  - Reduced code size but similar performance
  - Capable of handling all processing requirements in one operation state

Unified Assembler Language

- UAL is common syntax
  - Canonical form for all ARM and Thumb instructions
  - Code written using UAL can be assembled for ARM or Thumb for any ARM processor
    - For portability of the code
  - Assembler options are: --32, --arm, --thumb, or --Thumbx
- Many operations that would require two or more 16-bit instructions
  - Can be more efficiently executed with a single 32-bit instruction
- For GNU GAS, the directive for UAL is

  .syntax unified



PKH	QADD	QADD16	QADD8	QASX	QDADD	QDSUB	QSAX	QSUB
QSUB16	QSUB8	SADD16	SADD8	SASX	SEL	SHADD16	SHADD8	SHASX
SHSAX	SHSUB16	SHSUB8	SMLABB	SMLABT	SMLATB	SMLATT	SMLAD	SMLALBB

SMLALBT	SMLALTB
SMLALTT	SMLALD
SMLAWB	SMLAWT
SMLSD	SMLSLD
SMMLA	SMMLS
SMMUL	SMUAD
SMULBB	SMULBT
SMULTB	SMULTT
SMULWB	SMULWT
SMUSD	SSAT16
SSAX	SSUB16
SSUB8	SXTAB
SXTAB16	SXTAH
SXTB16	UADD16
UADD8	UASX
UHADD16	UHADD8
UHASX	UHSAX
UHSUB16	UHSUB8
UMAAL	UQADD16
UQADD8	UQASX
UQSAX	UQSUB16
UQSUB8	USAD8
USADA8	USAT16

USAX	USUB16	USUB8	UXTAB	UXTAB16	UXTAH	UXTB16

Cortex-M4

ADC	ADD	ADR	AND	ASR	B	CLZ
BFC	BFI	BIC	CDP	CLREX	CBNZ CBZ	CMN
CMP				DBG	EOR	LDC
LDMIA				LDMDB	LDR	LDRB
LDRBT				LDRD	LDREX	LDREXB
LDREXH				LDRH	LDRHT	LDRSB
LDRSBT				LDRSHT	LDRSH	LDRT
MCR				LSL	LSR	MLS
MCRR				MLA	MOV	MOVT
MRC				MRRC	MUL	MVN
NOP				ORN	ORR	PLD
PLDW				PLI	POP	PUSH
RBIT				REV	REV16	REVSH
ROR				RRX	RSB	SBC
SBFX				SDIV	SEV	SMLAL
SMULL				SSAT	STC	STMIA
STMDB				STR	STRB	STRBT
STRD	STREX	STREXB	STREXH	STRH	STRHT	STRT
SUB	SXTB	SXTH	TBB	TBH	TEQ	TST
UBFX	UDIV	UMLAL	UMULL	USAT	UXTB	UXTH
WFE	WFI	YIELD	IT			

CORTEX-M3

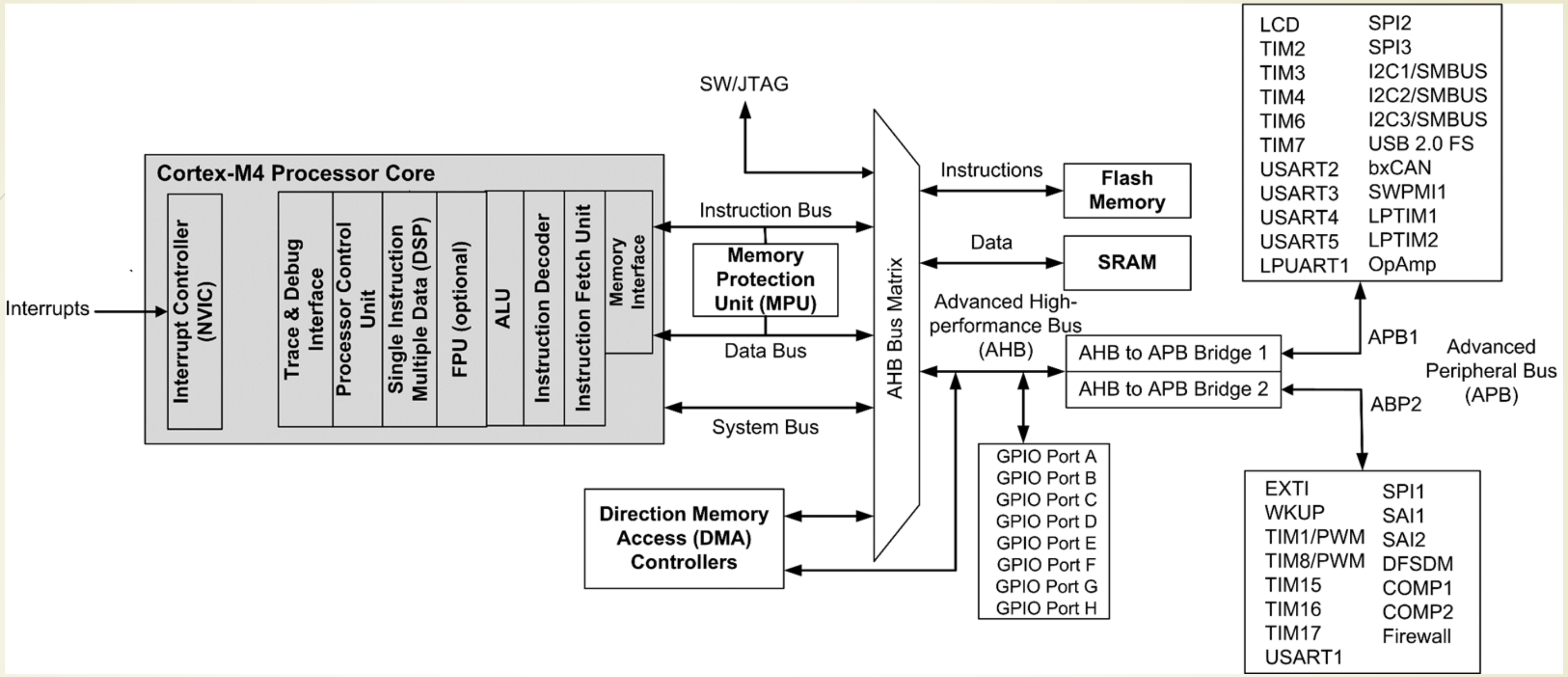
BKPT	BLX	ADC	ADD	ADR
BX	CPS	AND	ASR	B
DMB		BL		BIC
DSB		CMN	CMP	EOR
ISB		LDR	LDRB	LDM
MRS		LDRH	LDRSB	LDRSH
MSR		LSL	LSR	MOV
NOP	REV	MUL	MVN	ORR
REV16	REVSH	POP	PUSH	ROR
SEV	SXTB	RSB	SBC	STM
SXTH	UXTB	STR	STRB	STRH
UXTH	WFE	SUB	SVC	TST
WFI	YIELD			

CORTEX-M0/M1

VABS	VADD	VCMP	VCMPE	VCVT	VCVTR	VDIV	VLDM	VLDR
VMLA	VMLS	VMOV	VMRS	VMSR	VMUL	VNEG	VNMLA	VNMLS
VNMUL	VPOP	VPUSH	VSQRT	VSTM	VSTR	VSUB		

Cortex-M4F

Instruction Sets



ARM Cortex-M4 Organization (STM32L4)

CPU Bus

- Bus matrix allows concurrent data streams
  - Connects high speed components such as memory, DMA
- Advanced peripheral bus
  - Low bandwidth communication
  - Used for peripherals
  - Connected to Bus matrix via bridges.
- Advanced high performance Bus
  - High bandwidth communication
  - Used with memory and DMA

Instruction Set Architecture

- Accumulator based ISA
  - Pretty old
  - ALU source operand is implied and is in Accumulator
- Stack based ISA
  - Still old
  - ALU operands are assumed to be on top of stack
- Load Store ISA
  - Used in ARM
  - ALU source operands can be any registers
  - Only Load/Store instructions can access memory

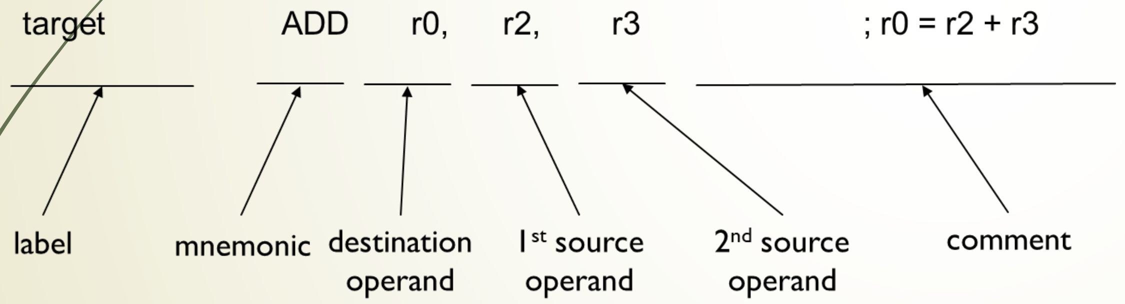
ARM Instruction Format

- label mnemonic operand1, operand2, operand3 ; comments

- Label is a reference to the memory address of this instruction.
- Mnemonic represents the operation to be performed.
- The number of operands varies, depending on each specific instruction. Some instructions have no operands at all.
  - Typically, operand1 is the destination register, and operand2 and operand3 are source operands.
  - operand2 is usually a register.
  - operand3 may be a register, an immediate number, a register shifted to a constant amount of bits, or a register plus an offset (used for memory access).
- Everything after the semicolon ";" is a comment, which is an annotation explicitly declaring programmers' intentions or assumptions.

ARM Instruction Format - Example

label mnemonic operand1, operand2, operand3 ; comments



Types of instructions supported

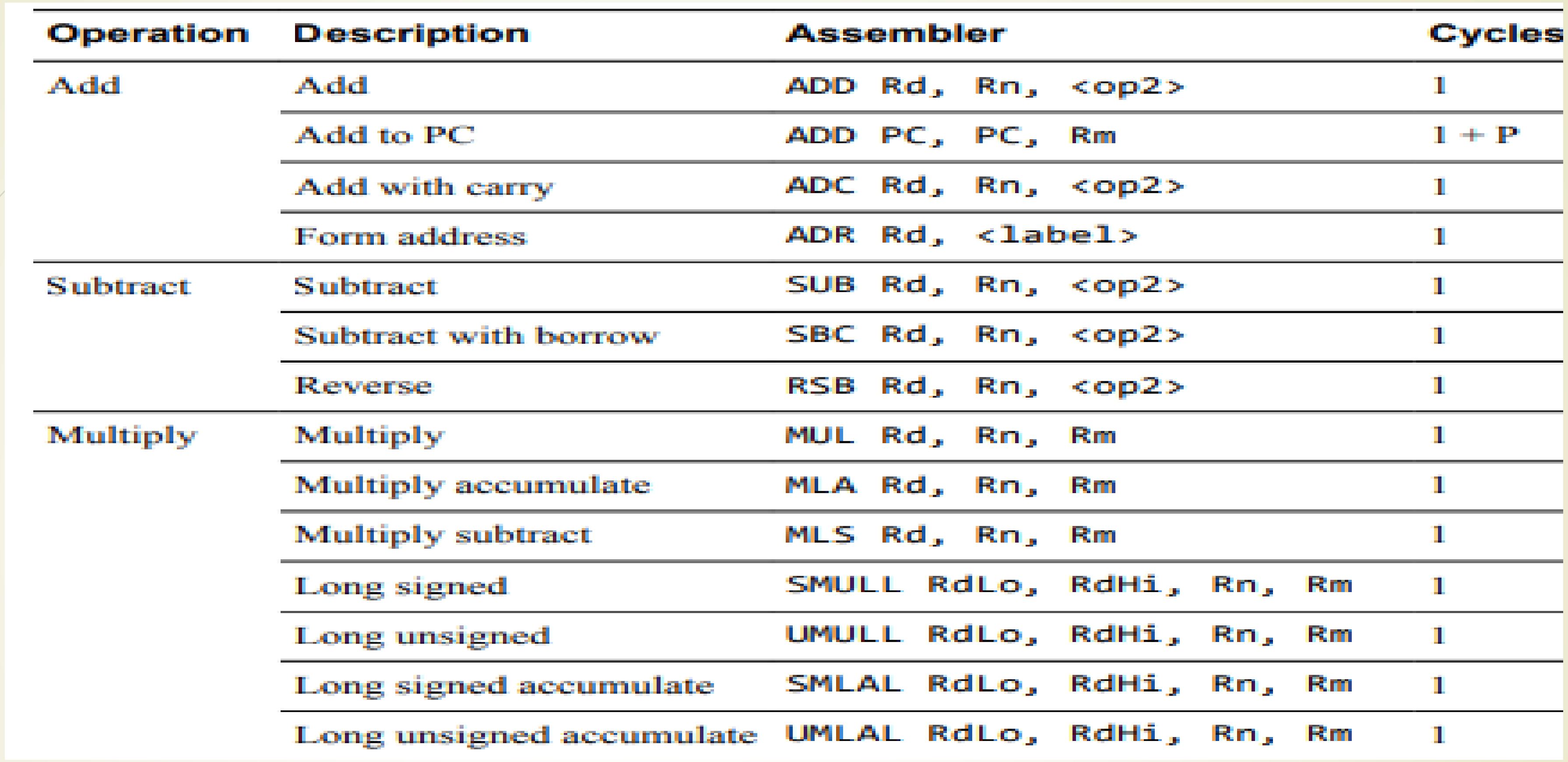
- Arithmetic and logic
  - Add, Subtract, Multiply, Divide, Shift, Rotate
- Data movement
  - Load, Store, Move
- Compare and branch
  - Compare, Test, If-then, Branch, compare and branch on zero
- Miscellaneous
  - Breakpoints, wait for events, interrupt enable/disable, data memory barrier, data synchronization barrier

Arithmetic and Logical

- Shift
  - LSL (logic shift left), LSR (logic shift right), ASR (arithmetic shift right), ROR (rotate right), RRX (rotate right with extend)
- Logic
  - AND (bitwise and), ORR (bitwise or), EOR (bitwise exclusive or), ORN (bitwise or not), MVN (move not)
- Bit set/clear
  - BFC (bit field clear), BFI (bit field insert), BIC (bit clear), CLZ (count leading zeroes)
- Bit/byte reordering
  - RBIT (reverse bit order in a word), REV (reverse byte order in a word), REV16 (reverse byte order in each half-word independently), REVSH (reverse byte order in each half-word independently)
- Addition
  - ADD, ADC (add with carry)
- Subtraction
  - SUB, RSB (reverse subtract), SBC (subtract with carry)

More Arithmetic and Logical

- Multiplication
  - MUL (multiply), MLA (multiply-accumulate), MLS (multiply-subtract), SMULL (signed long multiply-accumulate), SMLAL (signed long multiply-accumulate), UMULL (unsigned long multiply-subtract), UMLAL (unsigned long multiply-subtract)
- Division
  - SDIV (signed), UDIV (unsigned)
- Saturation
  - SSAT (signed), USAT (unsigned)
- Sign extension
  - SXTB (signed), SXTH, UXTB, UXTH
- Bit field extract
  - SBFX (signed), UBFX (unsigned)



Operation	Description	Assembler	Cycles
Add	Add	`ADD Rd, Rn, <op2>`	1
	Add to PC	`ADD PC, PC, Rm`	1 + P
	Add with carry	`ADC Rd, Rn, <op2>`	1
	Form address	`ADR Rd, <label>`	1
Subtract	Subtract	`SUB Rd, Rn, <op2>`	1
	Subtract with borrow	`SBC Rd, Rn, <op2>`	1
	Reverse	`RSB Rd, Rn, <op2>`	1
Multiply	Multiply	`MUL Rd, Rn, Rm`	1
	Multiply accumulate	`MLA Rd, Rn, Rm`	1
	Multiply subtract	`MLS Rd, Rn, Rm`	1
	Long signed	`SMULL RdLo, RdHi, Rn, Rm`	1
	Long unsigned	`UMULL RdLo, RdHi, Rn, Rm`	1
	Long signed accumulate	`SMLAL RdLo, RdHi, Rn, Rm`	1
	Long unsigned accumulate	`UMLAL RdLo, RdHi, Rn, Rm`	1

Examples

Add example

- Unified Assembler Language (UAL) Syntax

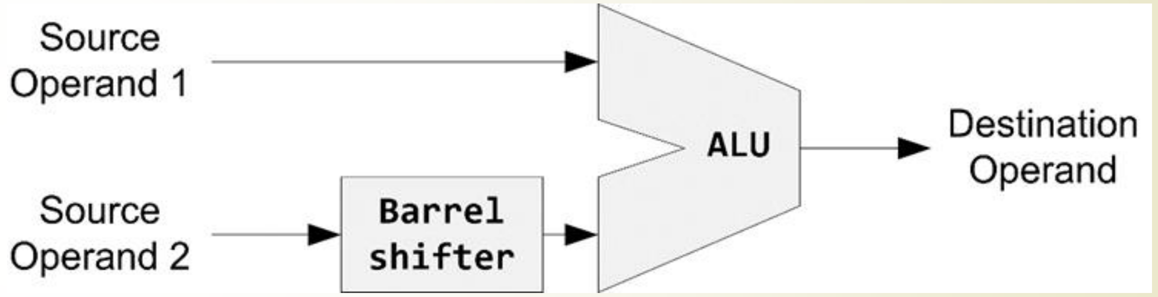
```
ADD r1, r2, r3    ; r1 = r2 + r3
ADD r1, r2, #4    ; r1 = r2 + 4
```

- Traditional Thumb Syntax

```
ADD r1, r3        ; r1 = r1 + r3
ADD r1, #15       ; r1 = r1 + 15
```

Barrel Shifter

- The second operand of ALU has a special hardware called Barrel shifter
- Barrel Shifter is a functional unit that is used to perform
  the Shift and Rotate Operations.
- Use Barrel shifter to speed up the application
- Example:
  - ADD r1, r0, r0, LSL #3 ; r1 = r0 + r0 << 3 = 9 × r0



BFC and BFI

- Bit Field Clear (BFC) and Bit Field Insert (BFI).
- Syntax

  BFC Rd, #lsb, #width

  BFI Rd, Rn, #lsb, #width

- Examples:

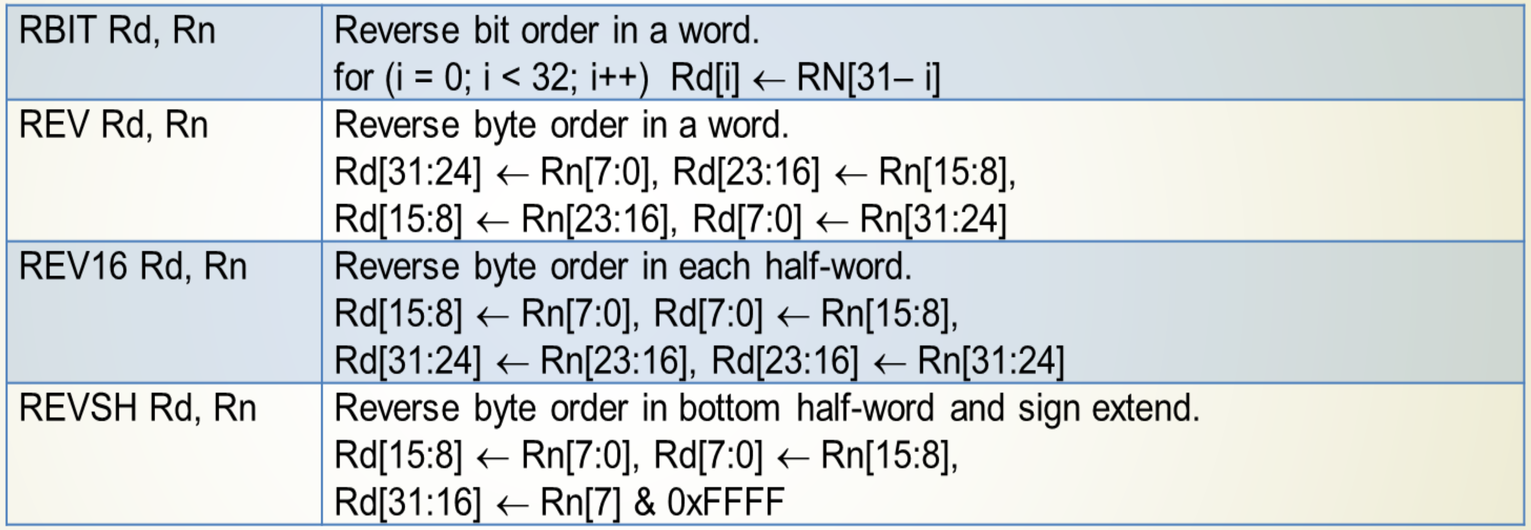
  BFC R4, #8, #12

  ; Clear bit 8 to bit 19 (12 bits) of R4 to 0

  BFI R9, R2, #8, #12

  ; Replace bit 8 to bit 19 (12 bits) of R9 with bit 0 to bit 11 from R2.

Reverse Order



RBIT Rd, Rn	Reverse bit order in a word. for (i = 0; i < 32; i++) Rd[i] ← RN[31– i]
REV Rd, Rn	Reverse byte order in a word. Rd[31:24] ← Rn[7:0], Rd[23:16] ← Rn[15:8], Rd[15:8] ← Rn[23:16], Rd[7:0] ← Rn[31:24]
REV16 Rd, Rn	Reverse byte order in each half-word. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:24] ← Rn[23:16], Rd[23:16] ← Rn[31:24]
REVSH Rd, Rn	Reverse byte order in bottom half-word and sign extend. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:16] ← Rn[7] & 0xFFFF

```
LDR  r0, =0x12345678 ; r0 = 0x12345678
RBIT r1, r0          ; Reverse bits, r1 = 0x1E6A2C48
REV16 R2, R0           ; R2 = 0x34127856
```

Examples

Suppose r0 = 0x56789ABC, find the result of the following operation.

```
RBIT  r1, r0
REV   r1, r0
REV16 r1, r0
REVSH r1, r0
```

R0 = 0x0101 0110 0111 1000 1001 1010 1011 1100

```
RBIT  r1, r0                ; 0011 1101 0101 1001 0001 1110 01010 1010
R1 = 0x3D591E6A
```

R0 = 0x0101 0110 0111 1000 1001 1010 1011 1100

```
REV   r1, r0                ; 1011 1100 1001 1010 0111 1000 0101 0110
R1 = 0xBC9A7856
```

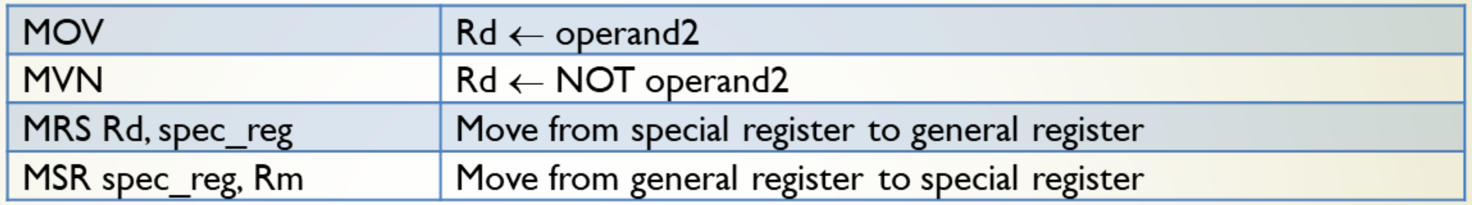
R0 = 0x0101 0110 0111 1000 1001 1010 1011 1100

```
REV16 r1, r0                ; 0111 1000 0101 0110 1011 1100 1001 1010
R1 = 0x7856BC9A
```

R0 = 0x0101 0110 0111 1000 1001 1010 1011 1100

```
REVSH r1, r0               ; 0xffff 1011 1100  1001 1010
R1 = 0xFFFFBC9A
```

Move Between Registers



MOV	Rd ← operand2
MVN	Rd ← NOT operand2
MRS Rd, spec_reg	Move from special register to general register
MSR spec_reg, Rm	Move from general register to special register

```
MOV r4, r5            ; Copy r5 to r4
MVN r4, r5            ; r4 = bitwise logical NOT of r5
MOV r1, r2, LSL #3    ; r1 = r2 << 3
MOV r0, PC            ; Copy PC (r15) to r0
MOV r1, SP            ; Copy SP (r14) to r1
```

Multiply examples

$$x = x * y + z - x;$$

Solution:

```
; r0 = x, r1 = y, r2 = z
MLA r2, r0, r1, r2    ;   r2 = r2 + r0 * r1,  z = x*y + z
SUB r0, r2, r0        ;   x = z - x
```

//Alternatively we could first multiply x and Y, subtract z and x and then add together

```
; r0 = x, r1 = y, r2 = z
MUL r1, r0, r1     ;   r1 = r0*r1 ; y = x *y
SUB r0, r2, r0        ;   x = z - x
Add r0, r0, r1          ; r0 = r0 + r1 ; x+y
```

Division examples

$x = x \% y\, ;\, x \text{ and } y \text{ are unsigned numbers}$

Solution:

```
; r0 = x, r1 = y
UDIV r2, r0, r1        ; r2 = floor(r0/r1)
MLS  r0, r1, r2, r0   ; r0 = r0 – r1*r2
```

Logical Operation examples

Lets assume , r0 = 0x0F0F0F0F, and r1 = 0xFEDCBA98

```
EOR r3, r1, r0          ; Exclusive OR
 R3 = 0xF1D3B597


 ORR r3, r1, r0           ; Simple OR
 R3 = 0xFFDFBF9F

 AND r3, r1, r0            ; Simple AND
 R3 = 0x0E0C0A08

 BIC r3, r1, r0              ; Bit clear, Logical AND NOT
 R3 = 0xF0D0B090
```

AREA	.section	Make a new block of data or code
ENTRY	armlink --entry=*location*	Declare an entry point where the program execution starts and command line for GNU
ALIGN	.ba**lign**	Align data or code to a particular memory boundary
DCB	.byte	Allocate one or more bytes (8 bits) of data
DCW	.hword	Allocate one or more half-words (16 bits) of data
DCD	.word	Allocate one or more words (32 bits) of data
SPACE	.org	Allocate a zeroed block of memory with a particular size
FILL	.fill	Allocate a block of memory and fill with a given value.
EQU	.equ	Give a symbol name to a numeric constant
RN	.req	Give a symbol name to a register
EXPORT	.global	Declare a symbol and make it referable by other source files
IMPORT	.global	Provide a symbol defined outside the current source file
INCLUDE/GET	.include	Include a separate source file within the current source file
PROC	function	Declare the start of a procedure
ENDP	.endp	Designate the end of a procedure
END	.end	Designate the end of a source file

Assembler Directive for ARM and GNU

ARM example

```
; Simple ARM syntax example
;
; Iterate round a loop 10 times, adding 1 to a register each time.

        AREA ||.text||, CODE, READONLY, ALIGN=2

main PROC
        MOV     r0,#0x64    ; r0 = 100
        MOV     r1,#0       ; r1 = 0
        B       test_loop   ; branch to test_loop
loop
        ADD     r0,r0,#1    ; Add 1 to r0
        ADD     r1,r1,#1    ; Add 1 to r1
test_loop
        CMP     r1,#0xa     ; if r1 < 10, branch back to loop
        BLT     loop
        ENDP

        END
```

GNU example

```
// Simple GNU syntax example
//
// Iterate round a loop 10 times, adding 1 to a register each time.

        .section .text,"x"
        .balign 4

main:
        MOV     r0,#0x64      // r0 = 100
        MOV     r1,#0         // r1 = 0
        B       test_loop     // branch to test_loop
loop:
        ADD     r0,r0,#1      // Add 1 to r0
        ADD     r1,r1,,#1     // Add 1 to r1
test_loop:
        CMP     wr14,#0xa     // if r1 < 10, branch back to loop
        BLT     loop
        .end
```

Bit-Band Operation

- Bit-band operation allows a single load/store operation to access a single bit in the memory
- For example, to change a single bit of one 32-bit data
  - Normal operation without bit-band (read-modify-write)
  - Read the value of 32-bit data
  - Modify a single bit of the 32-bit value (keep other bits unchanged)
  - Write the value back to the address
- Bit-band operation
  - Directly write a single bit (0 or 1) to the "bit-band alias address" of the data

Example

```
;Read-Modify-Write Operation

LDR R1, =0x20000000  ;Setup address
LDR R0, [R1]         ;Read
ORR.W R0, #0x8       ;Modify bit
STR R0, [R1]         ;Write back

;Bit-band Operation

LDR R1, =0x2200000C        ;Setup address
MOV R0, #1           ;Load data
STR R0, [R1]     ;Write
```

Cortex-M4 summary

Architecture	Armv7E-M
Bus Interface	3x AMBA AHB-Lite interface (Harvard bus architecture) AMBA ATB interface for CoreSight debug components
ISA Support	Thumb/Thumb-2
Pipeline	3-stage + branch speculation
DSP Extension	Single cycle 16/32-bit MAC Single cycle dual 16-bit MAC 8/16-bit SIMD arithmetic Hardware Divide (2-12 Cycles)
Floating-Point Unit	Optional single precision floating point unit IEEE 754 compliant
Memory Protection	Optional 8 region MPU with sub regions and background region
Bit Manipulation	Integrated Bit Field Processing Instructions & Bus Level Bit Banding
Interrupts	Non-maskable Interrupt (NMI) + 1 to 240 physical interrupts
Interrupt Priority Levels	8 to 256 priority levels
Wake-up Interrupt Controller	Optional
Sleep Modes	Integrated WFI and WFE Instructions and Sleep On Exit capability Sleep & Deep Sleep Signals Optional Retention Mode with Arm Power Management Kit

Further reading

- Cortex-M4 Technical Reference Manual: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439d/DDI0439D_cortex_m4_processor_r0p1_trm.pdf
- Cortex-M4 Devices Generic User Guide: http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf

- About the processor

https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4#:~:text=The%20Arm%20Cortex%2DM4%20processor,control%20and%20signal%20processing%20capabilities.

Next Lesson Topic

- Big and Little Endian
- Load and Store
- Flow Control
- Code examples
- C and Assembly