🧾 ARM Cortex-M3 Assembly Cheat Sheet — Directives (বাংলায় ব্যাখ্যা সহ)

🔢 Directive	🧠 অর্থ / কাজ	💬 বাংলা ব্যাখ্যা	💻 উদাহরণ
`.syntax unified`	Unified syntax সক্রিয় করে	ARM ও Thumb উভয় মোডের instruction একসাথে লেখার অনুমতি দেয়	`asm .syntax unified`
`.cpu cortex-m3`	টার্গেট প্রসেসর নির্ধারণ করে	বলে দেয় assembler-কে কোন CPU architecture এর instruction ব্যবহার করবে	`asm .cpu cortex-m3`
`.thumb`	Thumb মোডে কম্পাইল হবে	Cortex-M3 শুধুই Thumb instruction সাপোর্ট করে	`asm .thumb`
`.global label`	লেবেলকে গ্লোবাল করে	অন্য ফাইল থেকেও এই লেবেল (symbol) অ্যাক্সেস করা যাবে	`asm .global main`
`.section`	কোড বা ডেটা কোন সেকশনে যাবে বলে দেয়	যেমন `.text` (code), `.data` (initialized data), `.bss` (uninitialized data)	`asm .section .text`
`.text`	কোড সেকশন শুরু	সাধারণত কোড (instructions) এই সেকশনে থাকে	`asm .text`
`.data`	Initialized data সেকশন	যেমন ভ্যারিয়েবল যেগুলোর মান শুরুতেই দেওয়া আছে	`asm .data \n var1: .word 25`
`.bss`	Uninitialized data সেকশন	যেসব ভ্যারিয়েবল শুধু জায়গা নেবে, কিন্তু শুরুতে মান নেই	`asm .bss \n buffer: .space 32`
`.word`	32-bit data সংরক্ষণ	কোনো ভ্যারিয়েবল/constant ৪ বাইটে রাখা হয়	`asm num: .word 0x1234ABCD`

`.hword`	16-bit data	`asm val: .hword 0x1234`	
`.byte`	8-bit data	`asm flag: .byte 0xFF`	
`.space`	নির্দিষ্ট বাইট রিজার্ভ করে	RAM-এ জায়গা তৈরি কিন্তু মান দেওয়া হয় না	`asm buffer: .space 64`
`.align`	মেমরি alignment নির্ধারণ	Data বা instruction নির্দিষ্ট byte boundary তে রাখে	`asm .align 4`
`.equ name, value`	Constant define	C-এর `#define` এর মতো	`asm .equ LED_PIN, (1<<13)`
`.set name, value`	`.equ` এর মতোই, কিন্তু পরিবর্তন করা যায়		`asm .set COUNT, 10`
`.req`	Register alias define করে	কোনো রেজিস্টারকে নাম দেওয়া	`asm led_reg .req r1`
`.unreq`	`.req` এর নাম মুছে ফেলে	আর alias ব্যবহার করা যাবে না	`asm .unreq led_reg`
`.ltorg`	Literal pool রাখে	immediate মান বড় হলে assembler মেমরিতে রাখে, `.ltorg` সেই জায়গা তৈরি করে	`asm LDR r0, =0x20001000 \n .ltorg`
`.end`	ফাইলের শেষ নির্দেশ করে		`asm .end`
`.balign n`	নির্দিষ্ট boundary তে align করে	`asm .balign 4`	
`.ascii`	string (without null) সংরক্ষণ	`asm msg: .ascii "Hello"`	
`.asciz`	string + null byte সংরক্ষণ	`asm msg: .asciz "Hello\n"`	
`.pool`	Literal pool ঘোষণা করে	LDR pseudo-instruction এর জন্য immediate data রাখে	`asm .pool`

⚙️ ছোট উদাহরণ (সব একসাথে)

```
   .syntax unified
    .cpu cortex-m3
    .thumb
    .global main

    .data
msg:    .asciz "LED ON\n"
ledVal: .word 0x20000000

    .text
main:
    LDR r0, =msg            @ মেসেজের ঠিকানা
    LDR r1, =ledVal         @ LED ভ্যালুর ঠিকানা
    LDR r2, [r1]            @ LED ভ্যালু পড়া
    ADD r2, r2, #1
    STR r2, [r1]            @ নতুন মান লেখা
    BX lr                   @ ফিরে যাওয়া
    .end
```

💡 দ্রুত মনে রাখার টিপস

ক্যাটাগরি	নির্দেশ
কোড সেকশন	`.text`, `.global`, `.thumb`, `.cpu`, `.syntax`
ডেটা সেকশন	`.data`, `.bss`, `.word`, `.byte`, `.hword`, `.space`
কনস্ট্যান্ট	`.equ`, `.set`
নাম (alias)	`.req`, `.unreq`
এলাইনমেন্ট	`.align`, `.balign`
লিটারাল পুল	`.ltorg`, `.poo`

🧠 ARM Cortex-M3 Assembly Directives Cheat Sheet (বাংলায়)

Directive	কাজ (বাংলায় ব্যাখ্যা)	উদাহরণ ও ব্যাখ্যা
.syntax unified	ARM ও Thumb উভয়ের জন্য একীভূত সিনট্যাক্স ব্যবহার করে। CubeIDE বা Keil এ আধুনিক কোডে এটা অবশ্যই দিতে হয়।	`asm .syntax unified` 👉 এর মানে হলো ARM/Thumb দুইটাই unified সিনট্যাক্সে লেখা যাবে (MOV, ADD ইত্যাদি)।
.cpu cortex-m3	কোন CPU এর জন্য কোড লেখা হচ্ছে তা নির্ধারণ করে।	`asm .cpu cortex-m3` 👉 এটা দিলে assembler বুঝবে Cortex-M3 instruction সেট ব্যবহার করতে হবে।
.thumb	কোডটিকে Thumb মোডে (১৬-বিট instruction set) কম্পাইল করতে বলে। Cortex-M3 কেবল Thumb-2 মোডে কাজ করে, তাই এটা সবসময় দিতে হয়।	`asm .thumb`
.global	কোনো লেবেল বা ফাংশনকে গ্লোবাল (অন্য ফাইল থেকেও দেখা যাবে) করে।	`asm .global main` 👉 এখানে main ফাংশন অন্য ফাইল থেকেও কল করা যাবে।
.text	কোড (instruction) রাখার সেকশন শুরু করে।	`asm .text` 👉 এর নিচের অংশে প্রোগ্রামের instruction থাকবে।
.data	ডেটা (variable) রাখার সেকশন শুরু করে — যেগুলো RAM-এ লোড হয়।	`asm .data count: .word 5` 👉 `count` নামে একটা variable, যার মান ৫।
.bss	Uninitialized variable রাখার জন্য (যেগুলোর মান শুরুতে শূন্য থাকে)।	`asm .bss buffer: .space 20` 👉 `buffer` নামে ২০ বাইট জায়গা সংরক্ষণ হলো।
.word	৪-বাইট (৩২-বিট) ডেটা সংরক্ষণ করে।	`asm num: .word 0x12345678`
.byte	১-বাইট (৮-বিট) ডেটা সংরক্ষণ করে।	`asm flag: .byte 1`
.hword	২-বাইট (১৬-বিট) ডেটা সংরক্ষণ করে।	`asm val: .hword 0xABCD`

.asciz	ASCII string সংরক্ষণ করে এবং শেষে null (`\0`) দেয়।	`asm msg: .asciz "Hello ARM"` 👉 মেমরিতে থাকবে: `H e l l o A R M \0`
.space	নির্দিষ্ট পরিমাণ মেমরি স্পেস সংরক্ষণ করে (uninitialized)।	`asm arr: .space 10` 👉 ১০ বাইট ফাঁকা জায়গা।
.align	ডেটা মেমরি অ্যালাইনমেন্ট (যেমন ৪-বাইট বাউন্ডারি) ঠিক করে।	`asm .align 4` 👉 পরবর্তী ডেটা ৪-বাইট এলাইনে শুরু হবে।
.equ / .set	কোনো কনস্ট্যান্ট ভ্যালু নির্ধারণ করে।	`asm LED_PIN .equ 0x48000014` 👉 এখন `LED_PIN` মানে এই ঠিকানা।
.req	কোনো রেজিস্টারকে নাম দিয়ে সংজ্ঞায়িত করে (shortcut)।	`asm LED .req r0` 👉 এখন কোডে `LED` লিখলে `r0` বুঝবে।
.unreq	`.req` দিয়ে বানানো shortcut বাতিল করে।	`asm .unreq LED`
.ltorg	Literal pool (constant data যেমন immediate মান) এক্সপ্লিসিটলি রাখে, যাতে প্রোগ্রাম থেকে অ্যাক্সেস করা যায়।	সাধারণত বড় প্রোগ্রামে ব্যবহার হয়।
.end	অ্যাসেম্বলি ফাইলের শেষ নির্দেশ করে।	`asm .end`

🧩 উদাহরণ কোড: **LED Toggle (Cortex-M3 Assembly)**

```
.syntax unified
.cpu cortex-m3
.thumb
.global _start

.equ GPIO_ODR,  0x48000014     @ LED output register
.equ GPIO_IDR,  0x48000010     @ Button input register
.equ LED_PIN,   (1 << 5)
.equ BTN_PIN,   (1 << 13)
```

```
.text
_start:
    LDR r1, =GPIO_ODR
    LDR r2, =GPIO_IDR

loop:
    LDR r3, [r2]            @ Button input পড়া
    TST r3, #BTN_PIN
    BEQ off_led             @ যদি বোতাম না চাপা হয় → LED off

    LDR r4, [r1]
    ORR r4, r4, #LED_PIN
    STR r4, [r1]
    B loop

off_led:
    LDR r4, [r1]
    BIC r4, r4, #LED_PIN
    STR r4, [r1]
    B loop
.end
```

🪄 সারসংক্ষেপ (কেন দরকার)

ধরন	কাজ
`.syntax`, `.cpu`, `.thumb`	assembler কে CPU টাইপ বোঝায়
`.global`, `.text`	কোড সেকশন ও ফাংশন এক্সপোর্ট করে
`.data`, `.bss`, `.word`	ডেটা সংরক্ষণ
`.equ`, `.req`	কনস্ট্যান্ট ও নাম দেওয়া
`.align`, `.space`	মেমরি সংগঠিত করা

.end	কোডের সমাপ্তি