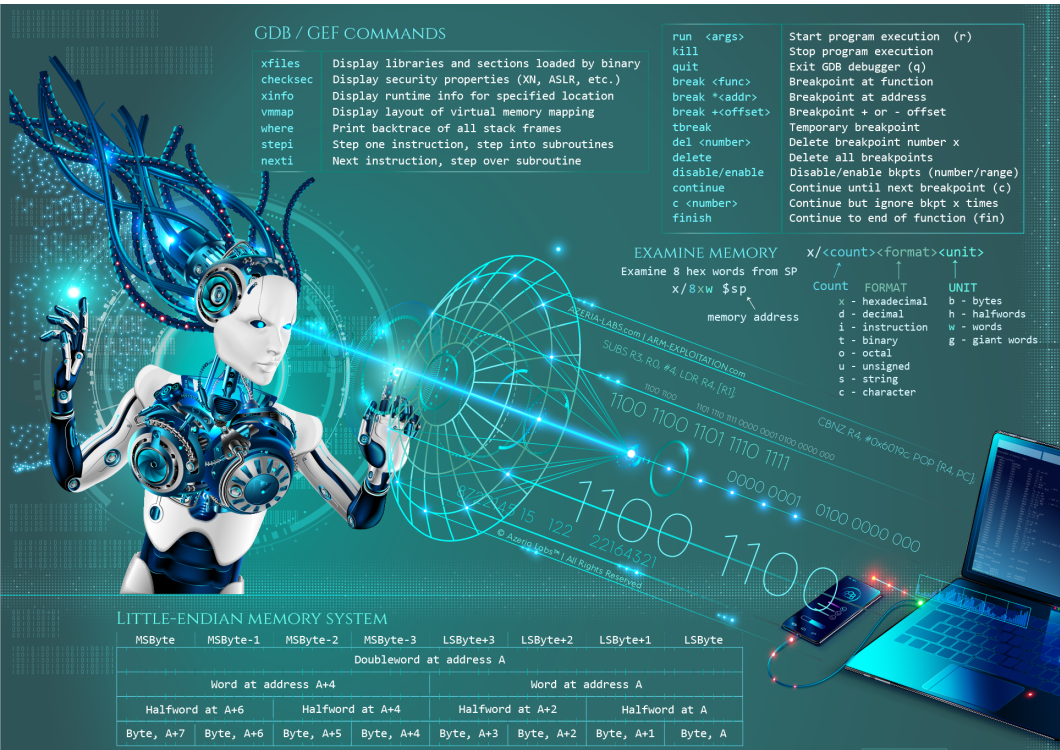


GDB / GEF COMMANDS

xfiles	Display libraries and sections loaded by binary
checksec	Display security properties (XN, ASLR, etc.)
xinfo	Display runtime info for specified location
vmmap	Display layout of virtual memory mapping
where	Print backtrace of all stack frames
stepi	Step one instruction, step into subroutines
nexti	Next instruction, step over subroutine

run <args>	Start program execution (r)
kill	Stop program execution
quit	Exit GDB debugger (q)
break <func>	Breakpoint at function
break *<addr>	Breakpoint at address
break +<offset>	Breakpoint + or - offset
tbreak	Temporary breakpoint
del <number>	Delete breakpoint number x
delete	Delete all breakpoints
disable/enable	Disable/enable bkpts (number/range)
continue	Continue until next breakpoint (c)
c <number>	Continue but ignore bkpt x times
finish	Continue to end of function (fin)

EXAMINE MEMORY

Examine 8 hex words from SP

```
x/8xw $sp
```

memory address

`x/<count><format><unit>`

Count	FORMAT	UNIT
	x - hexadecimal	b - bytes
	d - decimal	h - halfwords
	i - instruction	w - words
	t - binary	g - giant words
	o - octal	
	u - unsigned	
	s - string	
	c - character	

REGISTERS

R0	Argument & return value
R1 - R3	Arguments
R4 - R10	General purpose
R11	Frame Pointer
R12	Intra-Procedure-Call
SP	Stack Pointer
LR	Link Register
PC	Program Counter
CPSR	Current Program Status

R0 holds return value
R0-R3 hold arguments
If > 4 args: use stack

→ Keeps track of the stack frame
→ Points to the top of the stack
→ Holds the return address
→ Holds address of next instruction
→ Condition flags and other flags

CONDITION FLAGS

N	Z	C	V	Q	[...]

N = Negative flag
Z = Zero flag
C = Carry or borrow flag
V = Overflow flag
Q = Saturation flag

SET FLAGS

Execute instruction, update flags

MOVS mov, update flag
ADDS add, update flag
SUBS sub, update flag

Flags updated, regs unchanged

CMP compare
CMN compare negative
TEQ test equivalence
TST test bits

USE FLAGS

Example: Branch instructions

BEQ Branch if equal

```
loop:  CMP  r0, #4
       BEQ  func1   → func
       ADD  r0, r0, #1
       B    loop
```

CONDITION CODES

Code	Meaning	Flags tested
EQ	Equal (==)	Z == 1
NE	Not Equal (!=)	Z == 0
GT	Signed >	(Z==0)&&(N==V)
LT	Signed <	N != V
GE	Signed >=	N == V
LE	Signed <=	(Z==1)||(N!=V)
CS or HS	U. Higher or Same	C == 1
CC or LO	U. Lower	C == 0
MI	Negative -	N == 1
PL	Positive +	N == 0
AL	Always executed	-
NV	Never executed	-
VS	S. Overflow	V == 1
VC	No Overflow	V == 0
HI	U. Higher	(C==1)&&(Z==0)
LS	U. Lower or same	(C==0)||(Z==0)

BRANCHES

```
bx[cond] Rm

Thumb   ADD  r2, pc, #1
        BX   r2
        ADD  r1, r1
```

```
bl[cond] label        LR <- PC

MOV  r0, #2
MOV  r1, #4           func1
BL   func1            BX  lr
MOV  r2, #3
```

```
blx[cond] Rm          LR <- PC
blx       label       Thumb

MOV  r2, #2           func1
MOV  r1, #4           BX  lr
BLX  func1
MOV  r2, #3
```

PRIVILEGE MODES

User Mode	Application Code. Unprivileged: restricted access
Privileged	Used for OS Kernel, device drivers, boot code.
SVC	Supervisor: OS Kernel code.
FIQ	When high priority (fast) interrupt is raised
IRQ	When normal interrupt is raised
Undef	Used to handle undefined instructions
Abort	Used to handle memory access violations
System	Application code that requires privileges
Mon	Monitor: Gatekeeper between secure & non-secure states.
Hyp	Hypervisor: virtualization, only in non-secure state.

LITTLE-ENDIAN MEMORY SYSTEM

MSByte	MSByte-1	MSByte-2	MSByte-3	LSByte+3	LSByte+2	LSByte+1	LSByte
Doubleword at address A							
Word at address A+4				Word at address A			
Halfword at A+6		Halfword at A+4		Halfword at A+2		Halfword at A	
Byte, A+7	Byte, A+6	Byte, A+5	Byte, A+4	Byte, A+3	Byte, A+2	Byte, A+1	Byte, A

BIG-ENDIAN MEMORY SYSTEM

MSByte	MSByte-1	MSByte-2	MSByte-3	LSByte+3	LSByte+2	LSByte+1	LSByte
Doubleword at address A							
Word at address A				Word at address A+4			
Halfword at A		Halfword at A+2		Halfword at A+4		Halfword at A+6	
Byte, A	Byte, A+1	Byte, A+2	Byte, A+3	Byte, A+4	Byte, A+5	Byte, A+6	Byte, A+7

LITERAL POOL

The assembler uses literal pools to store some constant data in code sections.

Load Register (literal)
1. calculates address from PC value plus immediate offset
2. loads a word from memory
3. writes it to a register

```
LDR  R1, [PC, #16]
[...]
.word  0x000001ff
```

Memory (4 byte view)

Patch string /bin/shX → /bin/sh\0

assembly view
`.ascii "/bin/shX"`

disassembly view
`.word 0x6e69622f`
`.word 0x5868732f`

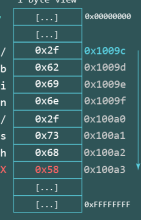
Memory (4 byte view)

Store Byte Instruction:

```
STRB  R2, [R0, #7]
```

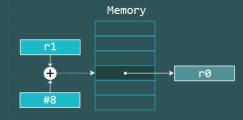
store byte from R2 to [address] in R0 + #7

Memory (1 byte view)



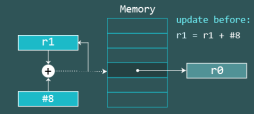
ADDRESSING MODES

Offset
`LDR r0, [r1, #8]`



Post-indexed
`LDR r0, [r1], #8`

Pre-indexed
`LDR r0, [r1, #8]!`



LOAD AND STORE MULTIPLE

`LDM[xx] sp, {r0,r1,r2}`

IA Increment After
IB Increment Before
DA Decrement After
DB Decrement Before

LOAD AND STORE

Value at [address] in R2 is loaded into register R1

```
LDR  R1, [R2]
STR  R1, [R2]
```

Value in R1 is stored to [address] found in R2

ADC	Add with Carry	ERET	Exception Return	PLD	Preload Data
ADD	Add without Carry	HVC	Hypervisor Call	POP	Pop registers off stack
ADR	Form PC-relative Address	IT	If-Then Instruction	PUSH	Push registers onto stack
AND	Bitwise AND	LDM	Load Multiple Registers	RBIT	Reverse Bits
ASR	Arithmetic Shift Right	LDR	Load (with offset)	REV	Byte-Reverse Word
BFC	Bit Field Clear	LSL	Logical Shift Left	RFE	Return From Exception
BFI	Bit Field Insert	LSR	Logical Shift Right	ROR	Rotate Right
BIC	Bitwise Bit Clear	MLA	Multiply-Accumulate	RSB	Reverse Subtract without carry
BKPT	Breakpoint	MLS	Multiply-Subtract	RSC	Reverse Subtract with carry
B	Branch	MOV	Copy operand to destination	SBC	Subtract with Carry
BL	Branch with Link	MUL	Multipy	SDIV	Signed Divide
BLX	Branch with Link and Exchange	MVN	Bitwise NOT	SETEND	Set endianness bit in CPSR
BX	Branch with Exchange	NOP	No Operation	SMC	Secure Monitor Call
CBZ	Compare and Branch on Zero	ORR	Bitwise OR	SRS	Store Return State onto stack
CBNZ	Compare and Branch on Non-Zero				
CLZ	Count Leading Zeros				
CMP	Compare				
CMN	Compare Negative				
CPS	Change Processor State				
DMB	Data Memory Barrier				
EOR	Bitwise Exclusive OR				

STM	Store Multiple registers	LDC	Transfer Data, mem -> Coprocessor
STR	Store (with offset)	MCR	Move from register -> coprocessor
SUB	Subtract without carry	MRC	Move from coprocessor >- register
SVC	SuperVisor Call	STC	Transfer Data mem <-> Coprocessor
TEQ	Test Equivalence (EOR)		
TST	Test (AND)		
UDIV	Unsigned Divide		
WFE	Wait For Event hint		
WFI	Wait For Event Interrupt hint		
CDP	Coprocessor data operations		