

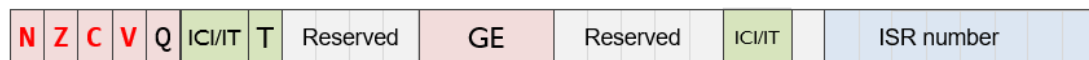
CSC111 Final Exam Cheatsheet

Decimal, Binary and Hex

Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

Condition Flags

Program Status Register (PSR)



Negative ----- signed result is negative

Zero ----- result is 0

Carry ----- add op → overflow
sub op doesn't borrow
last bit shifted out when shifting

oVerflow --- add/sub op → signed overflow

- ▶ **Negative bit**
 - ▶ $N = 1$ if most significant bit of result is 1
- ▶ **Zero bit**
 - ▶ $Z = 1$ if all bits of result are 0
- ▶ **Carry bit**
 - ▶ For unsigned addition, $C = 1$ if carry takes place
 - ▶ For unsigned subtraction, $C = 0$ (carry = not borrow) if borrow takes place
 - ▶ For shift/rotation, $C =$ last bit shifted out
- ▶ **oVerflow bit**
 - ▶ $V = 1$ if adding 2 same-signed numbers produces a result with the opposite sign
 - ▶ Positive + Positive = Negative, or
 - ▶ Negative + negative = Positive
 - ▶ Non-arithmetic operations does not touch V bit, such as MOV, AND, LSL, MUL

Carry and Overflow Flags w/ Arithmetic Instructions

Carry flag C = 1 (Borrow flag = 0) upon an **unsigned** addition if the answer is wrong (true result $> 2^n - 1$)

Carry flag C = 0 (Borrow flag = 1) upon an **unsigned** subtraction if the answer is wrong (true result < 0)

Overflow flag V = 1 upon a **signed** addition or subtraction if the answer is wrong (true result $> 2^{n-1} - 1$ or true result $< -2^{n-1}$)

Overflow may occur when adding 2 operands with the same sign, or subtracting 2 operands with different signs; Overflow cannot occur when adding 2 operands with different signs or when subtracting 2 operands with the same sign.

	Unsigned Addition	Unsigned Subtraction	Signed Addition or Subtraction
Carry flag	true result $> 2^n - 1 \rightarrow$ Carry flag=1 Borrow flag=0 (Result incorrect)	true result $< 0 \rightarrow$ Carry flag=0 Borrow flag=1 (Result incorrect)	N/A
Overflow flag	N/A	N/A	true result $> 2^{n-1} - 1$ or true result $< -2^{n-1}$ \rightarrow Overflow flag=1 (Result incorrect)

► 5

Condition Codes

Suffix	Description	Flags tested
EQ	E qual	Z=1
NE	N ot E qual	Z=0
CS/HS	Unsigned H igher or S ame	C=1
CC/LO	Unsigned L ower	C=0
MI	M inus (Negative)	N=1
PL	P lus (Positive or Zero)	N=0
VS	o verflow S et	V=1
VC	o verflow C leared	V=0
HI	Unsigned H igher	C=1 & Z=0
LS	Unsigned L ower or S ame	C=0 or Z=1
GE	Signed G reater or E qual	N=V
LT	Signed L ess T han	N!=V
GT	Signed G reater T han	Z=0 & N=V
LE	Signed L ess than or E qual	Z=1 or N!=V
AL	A lways	

► Condition Codes:

- EQ/NE: Z=1 / Z=0 (Equal/Not Equal)
- LT/GE: N≠V / N=V (Signed Less Than/Greater Equal)
- GT/LE: Z=0 & N=V / Z=1 or N≠V (Signed Greater/Less Equal)
- LO/HS: C=0 / C=1 (Unsigned Lower/Higher Same)
- HI/LS: C=1 & Z=0 / C=0 or Z=1 (Unsigned Higher/Lower Same)

► Flag Setting Instructions:

- CMP: R1 - R2 (result discarded)
- TST: R1 & R2 (result discarded)
- TEQ: R1 \oplus R2 (result discarded)

- CMN: R1 + R2 (result discarded)

Branch Instructions

	Instruction	Description	Flags tested
Unconditional Branch	B <i>Label</i>	Branch to label	
Conditional Branch	BEQ <i>Label</i>	Branch if E Qual	Z = 1
	BNE <i>Label</i>	Branch if N ot E qual	Z = 0
	BCS/BHS <i>Label</i>	Branch if unsigned H igher or S ame	C = 1
	BCC/BLO <i>Label</i>	Branch if unsigned L ower	C = 0
	BMI <i>Label</i>	Branch if M inus (Negative)	N = 1
	BPL <i>Label</i>	Branch if P lus (Positive or Zero)	N = 0
	BVS <i>Label</i>	Branch if o V erflow S et	V = 1
	BVC <i>Label</i>	Branch if o V erflow C lear	V = 0
	BHI <i>Label</i>	Branch if unsigned H igher	C = 1 & Z = 0
	BLS <i>Label</i>	Branch if unsigned L ower or S ame	C = 0 or Z = 1
	BGE <i>Label</i>	Branch if signed G reater or E qual	N = V
	BLT <i>Label</i>	Branch if signed L ess T han	N != V
	BGT <i>Label</i>	Branch if signed G reater T han	Z = 0 & N = V
	BLE <i>Label</i>	Branch if signed L ess than or E qual	Z = 1 or N = ! V

Ch16 Timer PWM Equations

- Timer clock frequency f_{CK_CNT} vs. CPU Clock Frequency f_{SOURCE}

$$f_{CK_CNT} = \frac{f_{SOURCE}}{PSC + 1}$$

- Timer interrupt frequency f_{Timer} with up-counting or down-counting mode:

$$f_{Timer} = \frac{f_{CK_CNT}}{ARR + 1}; \text{Timer Period} = \frac{ARR + 1}{f_{CK_CNT}} = (ARR + 1) * \text{Clock Period}$$

- Timer interrupt frequency f_{Timer} with center-aligned counting mode: $f_{Timer} = \frac{f_{CK_CNT}}{2 * ARR}; \text{Timer Period} = (2 * ARR) * \text{Clock Period}$

- PWM duty cycle for Mode 1 (Low-True):

$$\text{Duty Cycle} = \frac{CCR}{ARR + 1}$$

- PWM duty cycle for Mode 2 (High-True):

$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR + 1}$$

Cortex-M3 Assembly Instructions

Instruction	Operands	Description and Action
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry, $Rd \leftarrow Rn + Op2 + \text{Carry}$, ADCS updates N,Z,C,V
ADD, ADDS	{Rd,} Rn, Op2	Add, $Rd \leftarrow Rn + Op2$, ADDS updates N,Z,C,V
ADD, ADDS	{Rd,} Rn, #imm12	Add Immediate, $Rd \leftarrow Rn + \text{imm12}$, ADDS updates N,Z,C,V
ADR	Rd, label	Load PC-relative Address, $Rd \leftarrow \langle \text{label} \rangle$
AND, ANDS	{Rd,} Rn, Op2	Logical AND, $Rd \leftarrow Rn \text{ AND } Op2$, ANDS updates N,Z,C
ASR, ASRS	Rd, Rm, <Rs n>	Arithmetic Shift Right, $Rd \leftarrow Rm \gg (Rs n)$, ASRS updates N,Z,C
B	label	Branch, $PC \leftarrow \text{label}$
BFC	Rd, #lsb, #width	Bit Field Clear, $Rd[(\text{width}+\text{lsb}-1):\text{lsb}] \leftarrow 0$
BFI	Rd, Rn, #lsb, #width	Bit Field Insert, $Rd[(\text{width}+\text{lsb}-1):\text{lsb}] \leftarrow Rn[(\text{width}-1):0]$
BIC, BICS	{Rd,} Rn, Op2	Bit Clear, $Rd \leftarrow Rn \text{ AND NOT } Op2$, BICS updates N,Z,C
BKPT	#imm	Breakpoint, prefetch abort or enter debug state
BL	label	Branch with Link, $LR \leftarrow \text{address of next instruction}$, $PC \leftarrow \text{label}$
BLX	Rm	Branch register with link, $LR \leftarrow \text{address of next instruction}$, $PC \leftarrow Rm[31:1]$
BX	Rm	Branch register, $PC \leftarrow Rm$
CBNZ	Rn, label	Compare and Branch if Non-zero; $PC \leftarrow \text{label}$ if $Rn \neq 0$
CBZ	Rn, label	Compare and Branch if Zero; $PC \leftarrow \text{label}$ if $Rn == 0$
CLREX	-	Clear local processor exclusive tag
CLZ	Rd, Rm	Count Leading Zeroes, $Rd \leftarrow \text{number of leading zeroes in } Rm$
CMN	Rn, Op2	Compare Negative, Update N,Z,C,V flags on $Rn + Op2$
CMP	Rn, Op2	Compare, Update N,Z,C,V flags on $Rn - Op2$
CPSID	i	Disable specified (i) interrupts, optional change mode
CPSIE	i	Enable specified (i) interrupts, optional change mode
DMB	-	Data Memory Barrier, ensure memory access order
DSB	-	Data Synchronization Barrier, ensure completion of access
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR, $Rd \leftarrow Rn \text{ XOR } Op2$, EORS updates N,Z,C
ISB	-	Instruction Synchronization Barrier
IT	-	If-Then Condition Block
LDM	Rn{!}, reglist	Load Multiple Registers increment after, <reglist> = mem[Rn], Rn increments after each memory access
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple Registers Decrement Before, <reglist> = mem[Rn], Rn decrements before each memory access
LDMFD, LDMIA	Rn{!}, reglist	<reglist> = mem[Rn], Rn increments after each memory access
LDR	Rt, [Rn, #offset]	Load Register with Word, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with Byte, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two words, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$, $Rt2 \leftarrow \text{mem}[Rn + \text{offset} + 4]$
LDREX	Rt, [Rn, #offset]	Load Register Exclusive, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LDREXB	Rt, [Rn]	Load Register Exclusive with Byte, $Rt \leftarrow \text{mem}[Rn]$
LDREXH	Rt, [Rn]	Load Register Exclusive with Halfword, $Rt \leftarrow \text{mem}[Rn]$
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with Halfword, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with Signed Byte, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with Signed Halfword, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LDRT	Rt, [Rn, #offset]	Load Register with Word, $Rt \leftarrow \text{mem}[Rn + \text{offset}]$
LSL, LSLS	Rd, Rm, <Rs n>	Logic Shift Left, $Rd \leftarrow Rm \ll Rs n$, LSLS update N,Z,C
LSR, LSRS	Rd, Rm, <Rs n>	Logic Shift Right, $Rd \leftarrow Rm \gg Rs n$, LSRS update N,Z,C
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, $Rd \leftarrow (Ra + (Rn * Rm))[31:0]$
MLS	Rd, Rn, Rm, Ra	Multiply with Subtract, $Rd \leftarrow (Ra - (Rn * Rm))[31:0]$
MOV, MOVS	Rd, Op2	Move, $Rd \leftarrow Op2$, MOVS updates N,Z,C
MOVT	Rd, #imm16	Move Top, $Rd[31:16] \leftarrow \text{imm16}$, $Rd[15:0]$ unaffected
MOVW, MOVWS	Rd, #imm16	Move 16-bit Constant, $Rd \leftarrow \text{imm16}$, MOVWS updates N,Z,C
MRS	Rd, spec_reg	Move from Special Register, $Rd \leftarrow \text{spec_reg}$
MSR	spec_reg, Rm	Move to Special Register, $\text{spec_reg} \leftarrow Rm$, Updates N,Z,C,V
MUL, MULS	{Rd,} Rn, Rm	Multiply, $Rd \leftarrow (Rn * Rm)[31:0]$, MULS updates N,Z
MVN, MVNS	Rd, Op2	Move NOT, $Rd \leftarrow 0xFFFFFFFF \text{ EOR } Op2$, MVNS updates N,Z,C
NOP	-	No Operation
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT, $Rd \leftarrow Rn \text{ OR NOT } Op2$, ORNS updates N,Z,C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR, $Rd \leftarrow Rn \text{ OR } Op2$, ORRS updates N,Z,C
POP	reglist	Canonical form of LDM SP!, <reglist>
PUSH	reglist	Canonical form of STMDB SP!, <reglist>

RBIT	Rd, Rn	Reverse Bits, 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 a Halfword, 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 Low Halfword and sign extend, Rd[15:8]←Rn[7:0], Rd[7:0]←Rn[15:8], Rd[31:16]←Rn[7]*&FFFF
ROR, RORS	Rd, Rm, <Rs n>	Rotate Right, Rd ← ROR(Rm, Rs n), RORS updates N,Z,C
RRX, RRXS	Rd, Rm	Rotate Right with Extend, Rd ← RRX(Rm), RRXS updates N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract, Rd ← Op2 – Rn, RSBS updates N,Z,C,V
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry, Rd ← Rn-Op2-NOT(Carry), updates NZCV
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract, Rd[(width-1):0] = Rn[(width+lsb-1):lsb], Rd[31:width] = Replicate(Rn[width+lsb-1])
SDIV	{Rd,} Rn, Rm	Signed Divide, Rd ← Rn/Rm
SEV	-	Send Event
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate, RdHi,RdLo ← signed(RdHi,RdLo + Rn*Rm)
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply, RdHi,RdLo ← signed(Rn*Rm)
SSAT	Rd, #n, Rm{,shift #s}	Signed Saturate, Rd ← SignedSat((Rm shift s), n). Update Q
STM	Rn{!}, reglist	Store Multiple Registers
STMDB, STMEA	Rn{!}, reglist	Store Multiple Registers Decrement Before
STMFD, STMIA	Rn{!}, reglist	Store Multiple Registers Increment After
STR	Rt, [Rn, #offset]	Store Register with Word, mem[Rn+offset] = Rt
STRB, STRBT	Rt, [Rn, #offset]	Store Register with Byte, mem[Rn+offset] = Rt
STRD	Rt, Rt, [Rn,#offset]	Store Register with two Words, mem[Rn+offset] = Rt, mem[Rn+offset+4] = Rt2
STREX	Rd, Rt, [Rn,#offset]	Store Register Exclusive, If allowed, mem[Rn + offset] ← Rt, clear exclusive tag, Rd ← 0. Else Rd ← 1.
STREXB	Rd, Rt, [Rn]	Store Register Exclusive Byte, mem[Rn] ← Rt[15:0] or mem[Rn] ← Rt[7:0], clear exclusive tag, Rd ← 0. Else Rd ← 1
STREXH	Rd, Rt, [Rn]	Store Register Exclusive Halfword, mem[Rn] ← Rt[15:0] or mem[Rn] ← Rt[7:0], clear exclusive tag, Rd ← 0. Else Rd ← 1
STRH, STRHT	Rt, [Rn, #offset]	Store Halfword, mem[Rn + offset] ← Rt[15:0]
STRT	Rt, [Rn, #offset]	Store Register with Translation, mem[Rn + offset] = Rt
SUB, SUBS	{Rd,} Rn, Op2	Subtraction, Rd ← Rn – Op2, SUBS updates N,Z,C,V
SUB, SUBS	{Rd,} Rn, #imm12	Subtraction, Rd ← Rn-imm12, SUBS updates N,Z,C,V
SVC	#imm	Supervisor Call
SXTB	{Rd,} Rm {,ROR #n}	Sign Extend Byte, Rd ← SignExtend((Rm ROR (8*n))[7:0])
SXTH	{Rd,} Rm {,ROR #n}	Sign Extend Halfword, Rd ← SignExtend((Rm ROR (8*n))[15:0])
TBB	[Rn, Rm]	Table Branch Byte, PC ← PC+ZeroExtend(Memory(Rn+Rm,1)<<1)
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword, PC ← PC + ZeroExtend(Memory(Rn+Rm<<1, 2)<<1)
TEQ	Rn, Op2	Test Equivalence, Update N,Z,C,V on Rn EOR Operand2
TST	Rn, Op2	Test, Update N,Z,C,V on Rn AND Op2
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract, Rd[(width-1):0] = Rn[(width+lsb-1):lsb], Rd[31:width] = Replicate(0)
UDIV	{Rd,} Rn, Rm	Unsigned Divide, Rd ← Rn/Rm
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate, RdHi,RdLo ← unsigned(RdHi,RdLo + Rn*Rm)
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply, RdHi,RdLo ← unsigned(Rn*Rm)
USAT	Rd, #n, Rm{,shift #s}	Unsigned Saturate, Rd←UnsignedSat((Rm shift s),n), Update Q
UXTB	{Rd,} Rm {,ROR #n}	Unsigned Extend Byte, Rd ← ZeroExtend((Rm ROR (8*n))[7:0])
UXTH	{Rd,} Rm {,ROR #n}	Unsigned Extend Halfword, Rd ← ZeroExtend((Rm ROR (8*n))[15:0])
WFE	-	Wait For Event
WFI	-	Wait for Interrupt