THUMB Instruction Summary

The table below lists all THUMB mode instructions with clock cycles, affected CPSR flags, Format/chapter number, and description.
Only register R0..R7 can be used in thumb mode (unless R8-15,SP,PC are explicitly mentioned).

Logical Operations

Instruction	Cycles	Flags	Format	Expl.
MOV Rd,Imm8bit	1S	NZ--	3	Rd=nn
MOV Rd,Rs	1S	NZ00	2	Rd=Rs+0
MOV R0..14,R8..15	1S	----	5	Rd=Rs
MOV R8..14,R0..15	1S	----	5	Rd=Rs
MOV R15,R0..15	2S+1N	----	5	PC=Rs
MVN Rd,Rs	1S	NZ--	4	Rd=NOT Rs
AND Rd,Rs	1S	NZ--	4	Rd=Rd AND Rs
TST Rd,Rs	1S	NZ--	4	Void=Rd AND Rs
BIC Rd,Rs	1S	NZ--	4	Rd=Rd AND NOT Rs
ORR Rd,Rs	1S	NZ--	4	Rd=Rd OR Rs
EOR Rd,Rs	1S	NZ--	4	Rd=Rd XOR Rs
LSL Rd,Rs,Imm5bit	1S	NZc-	1	Rd=Rs SHL nn
LSL Rd,Rs	1S+1I	NZc-	4	Rd=Rd SHL (Rs AND 0FFh)
LSR Rd,Rs,Imm5bit	1S	NZc-	1	Rd=Rs SHR nn
LSR Rd,Rs	1S+1I	NZc-	4	Rd=Rd SHR (Rs AND 0FFh)
ASR Rd,Rs,Imm5bit	1S	NZc-	1	Rd=Rs SAR nn
ASR Rd,Rs	1S+1I	NZc-	4	Rd=Rd SAR (Rs AND 0FFh)
ROR Rd,Rs	1S+1I	NZc-	4	Rd=Rd ROR (Rs AND 0FFh)
NOP	1S	----	5	R8=R8

Carry flag affected only if shift amount is non-zero.

Arithmetic Operations and Multiply

Instruction	Cycles	Flags	Format	Expl.
ADD Rd,Rs,Imm3bit	1S	NZCV	2	Rd=Rs+nn
ADD Rd,Imm8bit	1S	NZCV	3	Rd=Rd+nn
ADD Rd,Rs,Rn	1S	NZCV	2	Rd=Rs+Rn
ADD R0..14,R8..15	1S	----	5	Rd=Rd+Rs
ADD R8..14,R0..15	1S	----	5	Rd=Rd+Rs
ADD R15,R0..15	2S+1N	----	5	PC=Rd+Rs
ADD Rd,PC,Imm8bit*4	1S	----	12	Rd=(($+4) AND NOT 2)+nn
ADD Rd,SP,Imm8bit*4	1S	----	12	Rd=SP+nn
ADD SP,Imm7bit*4	1S	----	13	SP=SP+nn
ADD SP,-Imm7bit*4	1S	----	13	SP=SP-nn
ADC Rd,Rs	1S	NZCV	4	Rd=Rd+Rs+Cy
SUB Rd,Rs,Imm3Bit	1S	NZCV	2	Rd=Rs-nn
SUB Rd,Imm8bit	1S	NZCV	3	Rd=Rd-nn
SUB Rd,Rs,Rn	1S	NZCV	2	Rd=Rs-Rn
SBC Rd,Rs	1S	NZCV	4	Rd=Rd-Rs-NOT Cy
NEG Rd,Rs	1S	NZCV	4	Rd=0-Rs
CMP Rd,Imm8bit	1S	NZCV	3	Void=Rd-nn
CMP Rd,Rs	1S	NZCV	4	Void=Rd-Rs
CMP R0-15,R8-15	1S	NZCV	5	Void=Rd-Rs
CMP R8-15,R0-15	1S	NZCV	5	Void=Rd-Rs
CMN Rd,Rs	1S	NZCV	4	Void=Rd+Rs
MUL Rd,Rs	1S+mI	NZx-	4	Rd=Rd*Rs

Jumps and Calls

Instruction	Cycles	Flags	Format	Expl.
B disp	2S+1N	----	18	PC=$+/-2048
BL disp	3S+1N	----	19	PC=$+/-4M, LR=$+5
B{cond=true} disp	2S+1N	----	16	PC=$+/-0..256
B{cond=false} disp	1S	----	16	N/A
BX R0..15	2S+1N	----	5	PC=Rs, ARM/THUMB (Rs bit0)

```
SWI Imm8bit          2S+1N      ----  17  PC=8, ARM SVC mode, LR=$+2
BKPT Imm8bit         ???        ----  17  ??? ARM9 Prefetch Abort
BLX disp             ???        ----  ??? ??? ARM9
BLX R0..R14          ???        ----  ??? ??? ARM9
POP {Rlist,}PC    (n+1)S+2N+1I  ----  14
MOV R15,R0..15       2S+1N      ----   5  PC=Rs
ADD R15,R0..15       2S+1N      ----   5  PC=Rd+Rs
```

The thumb BL instruction occupies two 16bit opcodes, 32bit in total.

Memory Load/Store

```
Instruction         Cycles    Flags Format Expl.
LDR  Rd,[Rb,5bit*4] 1S+1N+1I  ----  9   Rd = WORD[Rb+nn]
LDR  Rd,[PC,8bit*4] 1S+1N+1I  ----  6   Rd = WORD[PC+nn]
LDR  Rd,[SP,8bit*4] 1S+1N+1I  ---- 11   Rd = WORD[SP+nn]
LDR  Rd,[Rb,Ro]     1S+1N+1I  ----  7   Rd = WORD[Rb+Ro]
LDRB Rd,[Rb,5bit*1] 1S+1N+1I  ----  9   Rd = BYTE[Rb+nn]
LDRB Rd,[Rb,Ro]     1S+1N+1I  ----  7   Rd = BYTE[Rb+Ro]
LDRH Rd,[Rb,5bit*2] 1S+1N+1I  ---- 10   Rd = HALFWORD[Rb+nn]
LDRH Rd,[Rb,Ro]     1S+1N+1I  ----  8   Rd = HALFWORD[Rb+Ro]
LDSB Rd,[Rb,Ro]     1S+1N+1I  ----  8   Rd = SIGNED_BYTE[Rb+Ro]
LDSH Rd,[Rb,Ro]     1S+1N+1I  ----  8   Rd = SIGNED_HALFWORD[Rb+Ro]
STR  Rd,[Rb,5bit*4] 2N        ----  9   WORD[Rb+nn] = Rd
STR  Rd,[SP,8bit*4] 2N        ---- 11   WORD[SP+nn] = Rd
STR  Rd,[Rb,Ro]     2N        ----  7   WORD[Rb+Ro] = Rd
STRB Rd,[Rb,5bit*1] 2N        ----  9   BYTE[Rb+nn] = Rd
STRB Rd,[Rb,Ro]     2N        ----  7   BYTE[Rb+Ro] = Rd
STRH Rd,[Rb,5bit*2] 2N        ---- 10   HALFWORD[Rb+nn] = Rd
STRH Rd,[Rb,Ro]     2N        ----  8   HALFWORD[Rb+Ro]=Rd
PUSH {Rlist}{LR}    (n-1)S+2N ---- 14
POP  {Rlist}{PC}              ---- 14   (ARM9: with mode switch)
STMIA Rb!,{Rlist}   (n-1)S+2N ---- 15
LDMIA Rb!,{Rlist}   nS+1N+1I  ---- 15
```

THUMB Binary Opcode Format

This table summarizes the position of opcode/parameter bits for THUMB mode instructions, Format 1-19.

```
Form|_15|_14|_13|_12|_11|_10|_9_|_8_|_7_|_6_|_5_|_4_|_3_|_2_|_1_|_0_|
__1_|_0___0___0_|__Op___|_______Offset______|____Rs_____|____Rd_____|Shifted
__2_|_0___0___0___1___1_|_I,_Op_|___Rn/nn___|____Rs_____|____Rd_____|ADD/SUB
__3_|_0___0___1_|__Op___|____Rd_____|_____________Offset_____________|Immedi.
__4_|_0___1___0___0___0___0_|______Op_______|____Rs_____|____Rd_____|AluOp
__5_|_0___1___0___0___0___1_|__Op___|Hd_|Hs_|____Rs_____|____Rd_____|HiReg/BX
__6_|_0___1___0___0___1_|____Rd_____|_____________Word________________|LDR PC
__7_|_0___1___0___1_|__Op___|_0_|___Ro______|____Rb_____|____Rd_____|LDR/STR
__8_|_0___1___0___1_|__Op___|_1_|___Ro______|____Rb_____|____Rd_____|""H/SB/SH
__9_|_0___1___1_|__Op___|_______Offset______|____Rb_____|____Rd_____|""{B}
_10_|_1___0___0___0_|Op_|_______Offset______|____Rb_____|____Rd_____|""H
_11_|_1___0___0___1_|Op_|____Rd_____|_____________Word________________|"" SP
_12_|_1___0___1___0_|Op_|____Rd_____|_____________Word________________|ADD PC/SP
_13_|_1___0___1___1___0___0___0___0_|_S_|___________Word______________|ADD SP,nn
_14_|_1___0___1___1_|Op_|_1___0_|_R_|_____________Rlist_______________|PUSH/POP
_17_|_1___0___1___1___1___1___1___0_|___________User_Data____________|BKPT ARM9
_15_|_1___1___0___0_|Op_|____Rb_____|_____________Rlist_______________|STM/LDM
_16_|_1___1___0___1_|_____Cond______|_________Signed_Offset_________|B{cond}
_U__|_1___1___0___1___1___1___1___0_|_____________var_________________|UndefARM9
_17_|_1___1___0___1___1___1___1___1_|___________User_Data____________|SWI
_18_|_1___1___1___0___0_|___________________Offset_____________________|B
_19_|_1___1___1___0___1_|_____________________________var_____________|_0_|BLX.ARM9
_U__|_1___1___1___0___1_|_____________________________var_____________|_1_|UndefARM9
_19_|_1___1___1___1_|_H_|_____________Offset_Low/High_______________|BL,BLX
```

Further UNDEFS ??? ARM9?

```
1011 0001 xxxxxxxx (reserved)
```

```
1011 0x1x xxxxxxxx (reserved)
1011 10xx xxxxxxxx (reserved)
1011 1111 xxxxxxxx (reserved)
1101 1110 xxxxxxxx (free for user)
```

THUMB Opcodes: Register Operations (ALU, BX)

THUMB.1: move shifted register

```
15-13  Must be 000b for 'move shifted register' instructions
12-11  Opcode
         00b: LSL Rd,Rs,#Offset   (logical/arithmetic shift left)
         01b: LSR Rd,Rs,#Offset   (logical    shift right)
         10b: ASR Rd,Rs,#Offset   (arithmetic shift right)
         11b: Reserved (used for add/subtract instructions)
10-6   Offset                     (0-31)
5-3    Rs - Source register       (R0..R7)
2-0    Rd - Destination register  (R0..R7)
```

Example: LSL Rd,Rs,#nn ; Rd = Rs << nn ; ARM equivalent: MOVS Rd,Rs,LSL #nn
Zero shift amount is having special meaning (same as for ARM shifts), LSL#0 performs no shift (the the carry flag remains unchanged), LSR/ASR#0 are interpreted as LSR/ASR#32. Attempts to specify LSR/ASR#0 in source code are automatically redirected as LSL#0, and source LSR/ASR#32 is redirected as opcode LSR/ASR#0.
Execution Time: 1S
Flags: Z=zeroflag, N=sign, C=carry (except LSL#0: C=unchanged), V=unchanged.

THUMB.2: add/subtract

```
15-11  Must be 00011b for 'add/subtract' instructions
10-9   Opcode (0-3)
         0: ADD Rd,Rs,Rn   ;add register        Rd=Rs+Rn
         1: SUB Rd,Rs,Rn   ;subtract register   Rd=Rs-Rn
         2: ADD Rd,Rs,#nn  ;add immediate       Rd=Rs+nn
         3: SUB Rd,Rs,#nn  ;subtract immediate  Rd=Rs-nn
       Pseudo/alias opcode with Imm=0:
         2: MOV Rd,Rs      ;move (affects cpsr) Rd=Rs+0
8-6    For Register Operand:
         Rn - Register Operand (R0..R7)
       For Immediate Operand:
         nn - Immediate Value  (0-7)
5-3    Rs - Source register       (R0..R7)
2-0    Rd - Destination register  (R0..R7)
```

Return: Rd contains result, N,Z,C,V affected (including MOV).
Execution Time: 1S

THUMB.3: move/compare/add/subtract immediate

```
15-13  Must be 001b for this type of instructions
12-11  Opcode
         00b: MOV Rd,#nn      ;move     Rd   = #nn
         01b: CMP Rd,#nn      ;compare  Void = Rd - #nn
         10b: ADD Rd,#nn      ;add      Rd   = Rd + #nn
         11b: SUB Rd,#nn      ;subtract Rd   = Rd - #nn
10-8   Rd - Destination Register  (R0..R7)
7-0    nn - Unsigned Immediate    (0-255)
```

ARM equivalents for MOV/CMP/ADD/SUB are MOVS/CMP/ADDS/SUBS same format.
Execution Time: 1S
Return: Rd contains result (except CMP), N,Z,C,V affected (for MOV only N,Z).

THUMB.4: ALU operations

```
15-10  Must be 010000b for this type of instructions
9-6    Opcode (0-Fh)
```

```
          0: AND Rd,Rs     ;AND logical       Rd = Rd AND Rs
          1: EOR Rd,Rs     ;XOR logical       Rd = Rd XOR Rs
          2: LSL Rd,Rs     ;log. shift left   Rd = Rd << (Rs AND 0FFh)
          3: LSR Rd,Rs     ;log. shift right  Rd = Rd >> (Rs AND 0FFh)
          4: ASR Rd,Rs     ;arit shift right  Rd = Rd SAR (Rs AND 0FFh)
          5: ADC Rd,Rs     ;add with carry    Rd = Rd + Rs + Cy
          6: SBC Rd,Rs     ;sub with carry    Rd = Rd - Rs - NOT Cy
          7: ROR Rd,Rs     ;rotate right      Rd = Rd ROR (Rs AND 0FFh)
          8: TST Rd,Rs     ;test            Void = Rd AND Rs
          9: NEG Rd,Rs     ;negate            Rd = 0 - Rs
          A: CMP Rd,Rs     ;compare         Void = Rd - Rs
          B: CMN Rd,Rs     ;neg.compare     Void = Rd + Rs
          C: ORR Rd,Rs     ;OR logical        Rd = Rd OR Rs
          D: MUL Rd,Rs     ;multiply          Rd = Rd * Rs
          E: BIC Rd,Rs     ;bit clear         Rd = Rd AND NOT Rs
          F: MVN Rd,Rs     ;not               Rd = NOT Rs
  5-3    Rs - Source Register       (R0..R7)
  2-0    Rd - Destination Register  (R0..R7)
```

ARM equivalent for NEG would be RSBS.
Return: Rd contains result (except TST,CMP,CMN),
Affected Flags:

```
  N,Z,C,V for  ADC,SBC,NEG,CMP,CMN
  N,Z,C   for  LSL,LSR,ASR,ROR (carry flag unchanged if zero shift amount)
  N,Z,C   for  MUL on ARMv4 and below: carry flag destroyed
  N,Z     for  MUL on ARMv5 and above: carry flag unchanged
  N,Z     for  AND,EOR,TST,ORR,BIC,MVN
```

Execution Time:

```
  1S      for  AND,EOR,ADC,SBC,TST,NEG,CMP,CMN,ORR,BIC,MVN
  1S+1I   for  LSL,LSR,ASR,ROR
  1S+mI   for  MUL on ARMv4 (m=1..4; depending on MSBs of incoming Rd value)
  1S+mI   for  MUL on ARMv5 (m=3; fucking slow, no matter of MSBs of Rd value)
```

THUMB.5: Hi register operations/branch exchange

```
  15-10  Must be 010001b for this type of instructions
  9-8    Opcode (0-3)
           0: ADD Rd,Rs   ;add        Rd = Rd+Rs
           1: CMP Rd,Rs   ;compare  Void = Rd-Rs  ;CPSR affected
           2: MOV Rd,Rs   ;move       Rd = Rs
           2: NOP         ;nop        R8 = R8
           3: BX  Rs      ;jump       PC = Rs     ;may switch THUMB/ARM
           3: BLX Rs      ;call       PC = Rs     ;may switch THUMB/ARM (ARM9)
  7      MSBd - Destination Register most significant bit (or BL/BLX flag)
  6      MSBs - Source Register most significant bit
  5-3    Rs - Source Register        (together with MSBs: R0..R15)
  2-0    Rd - Destination Register   (together with MSBd: R0..R15)
```

Restrictions: For ADD/CMP/MOV, MSBs and/or MSBd must be set, ie. it is not allowed that both are cleared.
When using R15 (PC) as operand, the value will be the address of the instruction plus 4 (ie. $+4). Except for BX R15: CPU switches to ARM state, and PC is auto-aligned as (($+4) AND NOT 2).
For BX, MSBs may be 0 or 1, MSBd must be zero, Rd is not used/zero.
For BLX, MSBs may be 0 or 1, MSBd must be set, Rd is not used/zero.
For BX/BLX, when Bit 0 of the value in Rs is zero:

```
  Processor will be switched into ARM mode!
  If so, Bit 1 of Rs must be cleared (32bit word aligned).
  Thus, BX PC (switch to ARM) may be issued from word-aligned address
  only, the destination is PC+4 (ie. the following halfword is skipped).
```

BLX may not use R15. BLX saves the return address as LR=PC+3 (with thumb bit).
Using BLX R14 is possible (sets PC=Old_LR, and New_LR=retadr).
Assemblers/Disassemblers should use MOV R8,R8 as NOP (in THUMB mode).
Return: Only CMP affects CPSR condition flags!
Execution Time:

```
 1S     for ADD/MOV/CMP
 2S+1N  for ADD/MOV with Rd=R15, and for BX
```

THUMB Opcodes: Memory Load/Store (LDR/STR)

THUMB.6: load PC-relative (for loading immediates from literal pool)

```
  15-11  Must be 01001b for this type of instructions
  N/A    Opcode (fixed)
           LDR Rd,[PC,#nn]      ;load 32bit    Rd = WORD[PC+nn]
  10-8   Rd - Destination Register   (R0..R7)
  7-0    nn - Unsigned offset        (0-1020 in steps of 4)
```

The value of PC will be interpreted as (($+4) AND NOT 2).
Return: No flags affected, data loaded into Rd.
Execution Time: 1S+1N+1I

THUMB.7: load/store with register offset

```
  15-12  Must be 0101b for this type of instructions
  11-10  Opcode (0-3)
          0: STR  Rd,[Rb,Ro]   ;store 32bit data  WORD[Rb+Ro] = Rd
          1: STRB Rd,[Rb,Ro]   ;store  8bit data  BYTE[Rb+Ro] = Rd
          2: LDR  Rd,[Rb,Ro]   ;load  32bit data  Rd = WORD[Rb+Ro]
          3: LDRB Rd,[Rb,Ro]   ;load   8bit data  Rd = BYTE[Rb+Ro]
  9      Must be zero (0) for this type of instructions
  8-6    Ro - Offset Register              (R0..R7)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.8: load/store sign-extended byte/halfword

```
  15-12  Must be 0101b for this type of instructions
  11-10  Opcode (0-3)
          0: STRH Rd,[Rb,Ro]  ;store 16bit data          HALFWORD[Rb+Ro] = Rd
          1: LDSB Rd,[Rb,Ro]  ;load sign-extended 8bit   Rd = BYTE[Rb+Ro]
          2: LDRH Rd,[Rb,Ro]  ;load zero-extended 16bit  Rd = HALFWORD[Rb+Ro]
          3: LDSH Rd,[Rb,Ro]  ;load sign-extended 16bit  Rd = HALFWORD[Rb+Ro]
  9      Must be set (1) for this type of instructions
  8-6    Ro - Offset Register              (R0..R7)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.9: load/store with immediate offset

```
  15-13  Must be 011b for this type of instructions
  12-11  Opcode (0-3)
          0: STR  Rd,[Rb,#nn]  ;store 32bit data   WORD[Rb+nn] = Rd
          1: LDR  Rd,[Rb,#nn]  ;load  32bit data   Rd = WORD[Rb+nn]
          2: STRB Rd,[Rb,#nn]  ;store  8bit data   BYTE[Rb+nn] = Rd
          3: LDRB Rd,[Rb,#nn]  ;load   8bit data   Rd = BYTE[Rb+nn]
  10-6   nn - Unsigned Offset              (0-31 for BYTE, 0-124 for WORD)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.10: load/store halfword

```
  15-12  Must be 1000b for this type of instructions
  11     Opcode (0-1)
          0: STRH Rd,[Rb,#nn]  ;store 16bit data   HALFWORD[Rb+nn] = Rd
          1: LDRH Rd,[Rb,#nn]  ;load  16bit data   Rd = HALFWORD[Rb+nn]
```

```
  10-6   nn - Unsigned Offset              (0-62, step 2)
  5-3    Rb - Base Register                (R0..R7)
  2-0    Rd - Source/Destination Register  (R0..R7)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB.11: load/store SP-relative

```
  15-12  Must be 1001b for this type of instructions
  11     Opcode (0-1)
          0: STR  Rd,[SP,#nn]  ;store 32bit data   WORD[SP+nn] = Rd
          1: LDR  Rd,[SP,#nn]  ;load  32bit data   Rd = WORD[SP+nn]
  10-8   Rd - Source/Destination Register  (R0..R7)
  7-0    nn - Unsigned Offset              (0-1020, step 4)
```

Return: No flags affected, data loaded either into Rd or into memory.
Execution Time: 1S+1N+1I for LDR, or 2N for STR

THUMB Opcodes: Memory Addressing (ADD PC/SP)

THUMB.12: get relative address

```
  15-12  Must be 1010b for this type of instructions
  11     Opcode/Source Register (0-1)
          0: ADD  Rd,PC,#nn    ;Rd = (($+4) AND NOT 2) + nn
          1: ADD  Rd,SP,#nn    ;Rd = SP + nn
  10-8   Rd - Destination Register         (R0..R7)
  7-0    nn - Unsigned Offset              (0-1020, step 4)
```

Return: No flags affected, result in Rd.
Execution Time: 1S

THUMB.13: add offset to stack pointer

```
  15-8   Must be 10110000b for this type of instructions
  7      Opcode/Sign
          0: ADD  SP,#nn       ;SP = SP + nn
          1: ADD  SP,#-nn      ;SP = SP - nn
  6-0    nn - Unsigned Offset    (0-508, step 4)
```

Return: No flags affected, SP adjusted.
Execution Time: 1S

THUMB Opcodes: Memory Multiple Load/Store (PUSH/POP and LDM/STM)

THUMB.14: push/pop registers

```
  15-12  Must be 1011b for this type of instructions
  11     Opcode (0-1)
          0: PUSH {Rlist}{LR}   ;store in memory, decrements SP (R13)
          1: POP  {Rlist}{PC}   ;load from memory, increments SP (R13)
  10-9   Must be 10b for this type of instructions
  8      PC/LR Bit (0-1)
          0: No
          1: PUSH LR (R14), or POP PC (R15)
  7-0    Rlist - List of Registers (R7..R0)
```

In THUMB mode stack is always meant to be 'full descending', ie. PUSH is equivalent to 'STMFD/STMDB' and POP to 'LDMFD/LDMIA' in ARM mode.
Examples:

```
 PUSH {R0-R3}     ;push R0,R1,R2,R3
 PUSH {R0,R2,LR}  ;push R0,R2,LR
```

```
POP  {R4,R7}     ;pop R4,R7
POP  {R2-R4,PC}  ;pop R2,R3,R4,PC
```

Note: When calling to a sub-routine, the return address is stored in LR register, when calling further sub-routines, PUSH {LR} must be used to save higher return address on stack. If so, POP {PC} can be later used to return from the sub-routine.
POP {PC} ignores the least significant bit of the return address (processor remains in thumb state even if bit0 was cleared), when intending to return with optional mode switch, use a POP/BX combination (eg. POP {R3} / BX R3).
ARM9: POP {PC} copies the LSB to thumb bit (switches to ARM if bit0=0).
Return: No flags affected, SP adjusted, registers loaded/stored.
Execution Time: nS+1N+1I (POP), (n+1)S+2N+1I (POP PC), or (n-1)S+2N (PUSH).

THUMB.15: multiple load/store

```
15-12  Must be 1100b for this type of instructions
11     Opcode (0-1)
        0: STMIA Rb!,{Rlist}   ;store in memory, increments Rb
        1: LDMIA Rb!,{Rlist}   ;load from memory, increments Rb
10-8   Rb - Base register (modified) (R0-R7)
7-0    Rlist - List of Registers     (R7..R0)
```

Both STM and LDM are incrementing the Base Register.
The lowest register in the list (ie. R0, if it's in the list) is stored/loaded at the lowest memory address.
Examples:

```
STMIA R7!,{R0-R2}  ;store R0,R1,R2
LDMIA R0!,{R1,R5}  ;store R1,R5
```

Return: No flags affected, Rb adjusted, registers loaded/stored.
Execution Time: nS+1N+1I for LDM, or (n-1)S+2N for STM.

Strange Effects on Invalid Rlist's
Empty Rlist: R15 loaded/stored (ARMv4 only), and Rb=Rb+40h (ARMv4-v5).
Writeback with Rb included in Rlist: Store OLD base if Rb is FIRST entry in Rlist, otherwise store NEW base (STM/ARMv4), always store OLD base (STM/ARMv5), no writeback (LDM/ARMv4/ARMv5; at this point, THUMB opcodes work different than ARM opcodes).

THUMB Opcodes: Jumps and Calls

THUMB.16: conditional branch

```
15-12  Must be 1101b for this type of instructions
11-8   Opcode/Condition (0-Fh)
        0: BEQ label        ;Z=1         ;equal (zero) (same)
        1: BNE label        ;Z=0         ;not equal (nonzero) (not same)
        2: BCS/BHS label    ;C=1         ;unsigned higher or same (carry set)
        3: BCC/BLO label    ;C=0         ;unsigned lower (carry cleared)
        4: BMI label        ;N=1         ;negative (minus)
        5: BPL label        ;N=0         ;positive or zero (plus)
        6: BVS label        ;V=1         ;overflow (V set)
        7: BVC label        ;V=0         ;no overflow (V cleared)
        8: BHI label        ;C=1 and Z=0 ;unsigned higher
        9: BLS label        ;C=0 or Z=1  ;unsigned lower or same
        A: BGE label        ;N=V         ;greater or equal
        B: BLT label        ;N<>V        ;less than
        C: BGT label        ;Z=0 and N=V ;greater than
        D: BLE label        ;Z=1 or N<>V ;less or equal
        E: Undefined, should not be used
        F: Reserved for SWI instruction (see SWI opcode)
7-0    Signed Offset, step 2 ($+4-256..$+4+254)
```

Destination address must by halfword aligned (ie. bit 0 cleared)
Return: No flags affected, PC adjusted if condition true

Execution Time:

```
  2S+1N   if condition true (jump executed)
  1S      if condition false
```

BX and ADD/MOV PC

See also THUMB.5: BX Rs, and ADD/MOV PC,Rs.

THUMB.18: unconditional branch

```
  15-11  Must be 11100b for this type of instructions
  N/A    Opcode (fixed)
          B label   ;branch (jump)
  10-0   Signed Offset, step 2 ($+4-2048..$+4+2046)
```

Return: No flags affected, PC adjusted.
Execution Time: 2S+1N

THUMB.19: long branch with link

This may be used to call (or jump) to a subroutine, return address is saved in LR (R14).
Unlike all other THUMB mode instructions, this instruction occupies 32bit of memory which are split into two 16bit THUMB opcodes.

```
 First Instruction - LR = PC+4+(nn SHL 12)
  15-11  Must be 11110b for BL/BLX type of instructions
  10-0   nn - Upper 11 bits of Target Address
 Second Instruction - PC = LR + (nn SHL 1), and LR = PC+2 OR 1 (and BLX: T=0)
  15-11  Opcode
          11111b: BL label   ;branch long with link
          11101b: BLX label  ;branch long with link switch to ARM mode (ARM9)
  10-0   nn - Lower 11 bits of Target Address (BLX: Bit0 Must be zero)
```

The destination address range is (PC+4)-400000h..+3FFFFEh, ie. PC+/-4M.
Target must be halfword-aligned. As Bit 0 in LR is set, it may be used to return by a BX LR instruction (keeping CPU in THUMB mode).
Return: No flags affected, PC adjusted, return address in LR.
Execution Time: 3S+1N (first opcode 1S, second opcode 2S+1N).
Note: Exceptions may or may not occur between first and second opcode, this is "implementation defined" (unknown how this is implemented in GBA and NDS).
Using only the 2nd half of BL as "BL LR+imm" is possible (for example, Mario Golf Advance Tour for GBA uses opcode F800h as "BL LR+0").

THUMB.17: software interrupt and breakpoint

SWI supposed for calls to the operating system - Enter Supervisor mode (SVC) in ARM state. BKPT intended for debugging - enters Abort mode in ARM state via Prefetch Abort vector.

```
  15-8   Opcode
          11011111b: SWI nn   ;software interrupt
          10111110b: BKPT nn  ;software breakpoint (ARMv5 and up)
  7-0    nn - Comment Field, ignored by processor (8bit value) (0-255)
```

Execution Time: 2S+1N
The exception handler may interprete the SWI Comment Field by examining the lower 8bit of the 16bit opcode opcode at [R14_svc-2].
If your are also using SWI's from inside of ARM mode, then the SWI handler must examine the T Bit SPSR_svc in order to determine whether it's been a ARM SWI - and if so, examine the lower 24bit of the 32bit opcode opcode at [R14_svc-4].
For Returning from SWI use "MOVS PC,R14", that instruction does restore both PC and CPSR, ie. PC=R14_svc, and CPSR=SPSR_svc, and (as called from THUMB mode), it'll also restore THUMB mode.
Nesting SWIs: SPSR_svc and R14_svc should be saved on stack before either invoking nested SWIs, or (if the IRQ handler uses SWIs) before enabling IRQs.
Execution SWI/BKPT:

```
  R14_svc=PC+2     R14_abt=PC+4   ;save return address
  SPSR_svc=CPSR    SPSR_abt=CPSR  ;save CPSR flags
```

```
  CPSR=<changed>   CPSR=<changed> ;Enter svc/abt, ARM state, IRQs disabled
  PC=VVVV0008h     PC=VVVV000Ch   ;jump to SWI/PrefetchAbort vector address
```