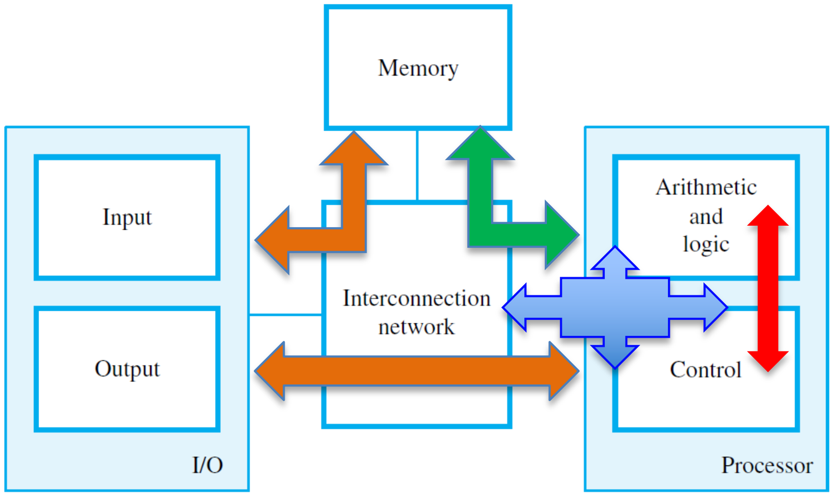
2.3 Instructions and Sequencing

- Instructions categories:
  - **Data transfer: memory to/from processor**
  - **Input/output transfer: I/O to/from processor/memory**
  - **Arithmetic and logic operations**
  - **Program sequencing and control**
  - ARM: 57 instructions (PM31-34)
  - Thumb: 36 instructions (TI5-2/3)



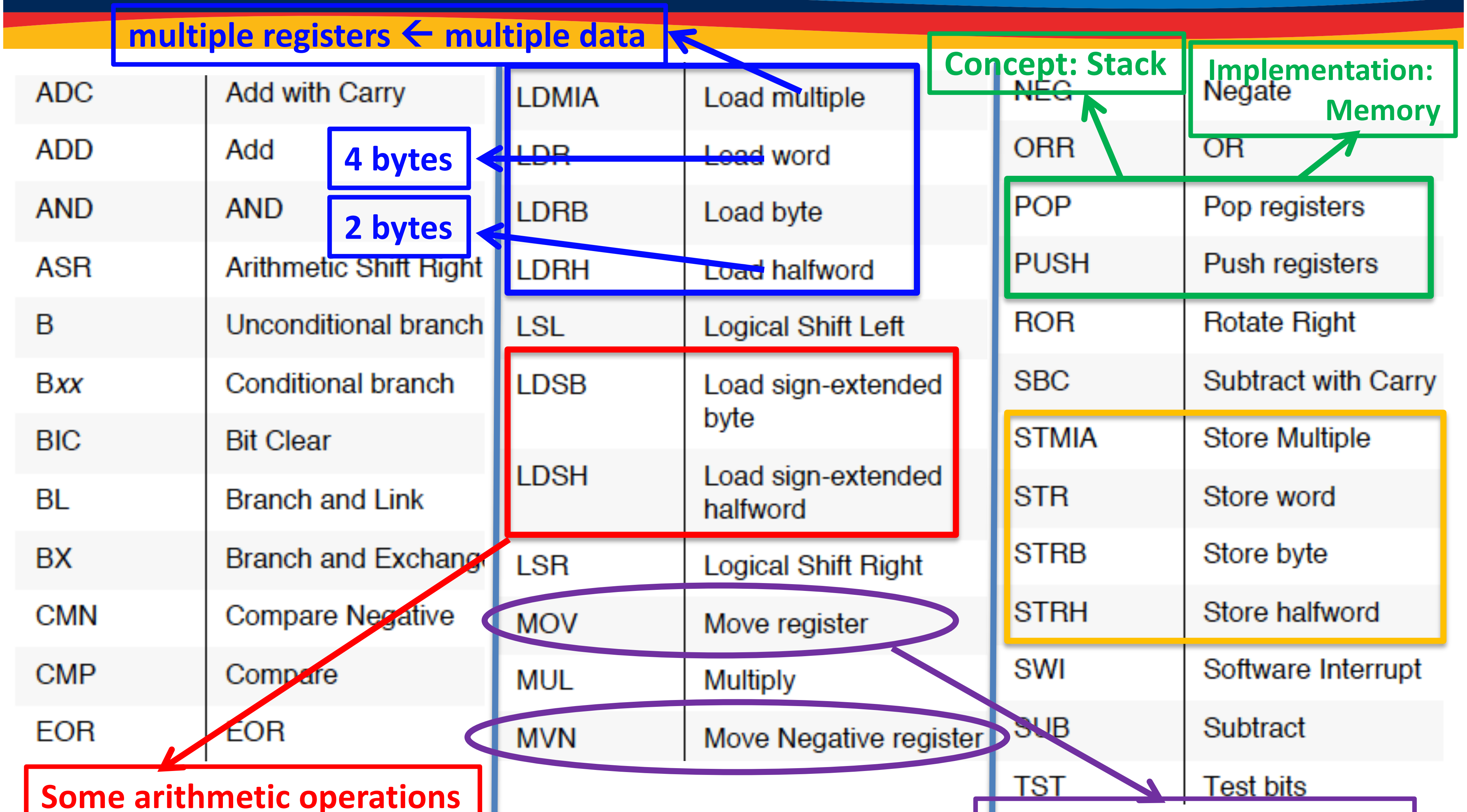
ARM Thumb Instructions (TH3-4)

ADC	Add with Carry	LDMIA	Load multiple	NEG	Negate
ADD	Add	LDR	Load word	ORR	OR
AND	AND	LDRB	Load byte	POP	Pop registers
ASR	Arithmetic Shift Right	LDRH	Load halfword	PUSH	Push registers
B	Unconditional branch	LSL	Logical Shift Left	ROR	Rotate Right
B*xx*	Conditional branch	LDSB	Load sign-extended byte	SBC	Subtract with Carry
BIC	Bit Clear	LDSH	Load sign-extended halfword	STMIA	Store Multiple
BL	Branch and Link	LSR	Logical Shift Right	STR	Store word
BX	Branch and Exchange	MOV	Move register	STRB	Store byte
CMN	Compare Negative	MUL	Multiply	STRH	Store halfword
CMP	Compare	MVN	Move Negative register	SWI	Software Interrupt
EOR	EOR			SUB	Subtract
				TST	Test bits

2.3.4 Load/Store Architecture

- Two operations to access (read or write) memory
  - **Load** : register ← memory
  - **Store** : memory ← register
  - Source: no change
  - Destination: overwritten
- ALU ← operands (only from registers; register size?)
- A characteristic of the Reduced Instruction Set Computer (RISC)
  - Storing data operand in registers: advantages ?

Thumb Memory Access Instructions



multiple registers ← multiple data

4 bytes

2 bytes

Concept: Stack

Implementation: Memory

Some arithmetic operations

Ri ← #imm or Ri ← Rj

ADC	Add with Carry
ADD	Add
AND	AND
ASR	Arithmetic Shift Right
B	Unconditional branch
Bxx	Conditional branch
BIC	Bit Clear
BL	Branch and Link
BX	Branch and Exchange
CMN	Compare Negative
CMP	Compare
EOR	EOR
LDMIA	Load multiple
LDR	Load word
LDRB	Load byte
LDRH	Load halfword
LSL	Logical Shift Left
LDSB	Load sign-extended byte
LDSH	Load sign-extended halfword
LSR	Logical Shift Right
MOV	Move register
MUL	Multiply
MVN	Move Negative register
NEG	Negate
ORR	OR
POP	Pop registers
PUSH	Push registers
ROR	Rotate Right
SBC	Subtract with Carry
STMIA	Store Multiple
STR	Store word
STRB	Store byte
STRH	Store halfword
SWI	Software Interrupt
SUB	Subtract
TST	Test bits

Thumb Data Processing Instructions (TH3-4)

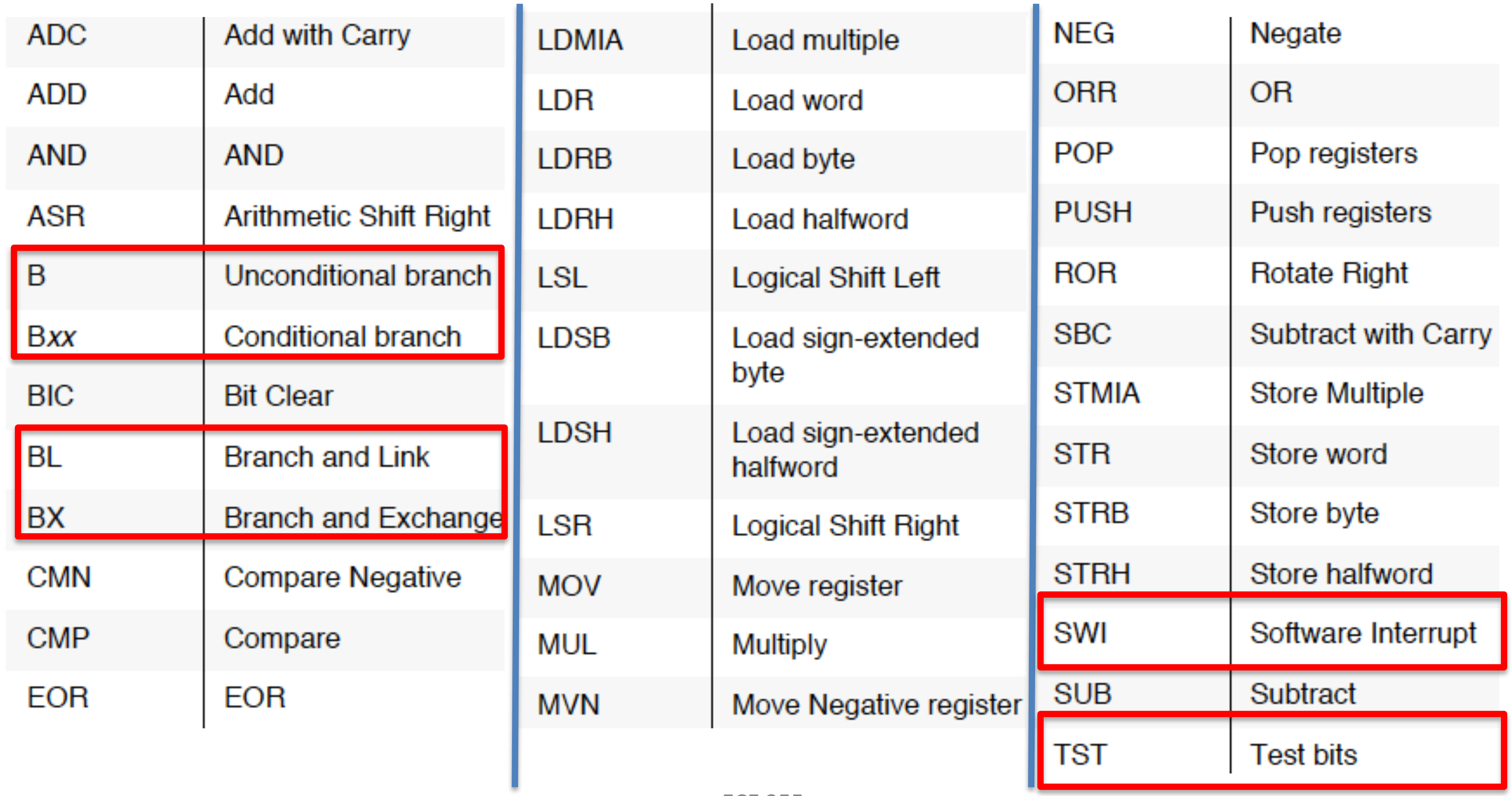
Arithmetic in Red

Logic in Green

Bit Operation in Blue

ADC	Add with Carry
ADD	Add
AND	AND
ASR	Arithmetic Shift Right
B	Unconditional branch
B*xx*	Conditional branch
BIC	Bit Clear
BL	Branch and Link
BX	Branch and Exchange
CMN	Compare Negative
CMP	Compare
EOR	EOR
LDMIA	Load multiple
LDR	Load word
LDRB	Load byte
LDRH	Load halfword
LSL	Logical Shift Left
LDSB	Load sign-extended byte
LDSH	Load sign-extended halfword
LSR	Logical Shift Right
MOV	Move register
MUL	Multiply
MVN	Move Negative register
NEG	Negate
ORR	OR
POP	Pop registers
PUSH	Push registers
ROR	Rotate Right
SBC	Subtract with Carry
STMIA	Store Multiple
STR	Store word
STRB	Store byte
STRH	Store halfword
SWI	Software Interrupt
SUB	Subtract
TST	Test bits

Thumb Branch/Control Instructions (TH3-4)



ADC	Add with Carry
ADD	Add
AND	AND
ASR	Arithmetic Shift Right
B	Unconditional branch
B*xx*	Conditional branch
BIC	Bit Clear
BL	Branch and Link
BX	Branch and Exchange
CMN	Compare Negative
CMP	Compare
EOR	EOR

LDMIA	Load multiple
LDR	Load word
LDRB	Load byte
LDRH	Load halfword
LSL	Logical Shift Left
LDSB	Load sign-extended byte
LDSH	Load sign-extended halfword
LSR	Logical Shift Right
MOV	Move register
MUL	Multiply
MVN	Move Negative register

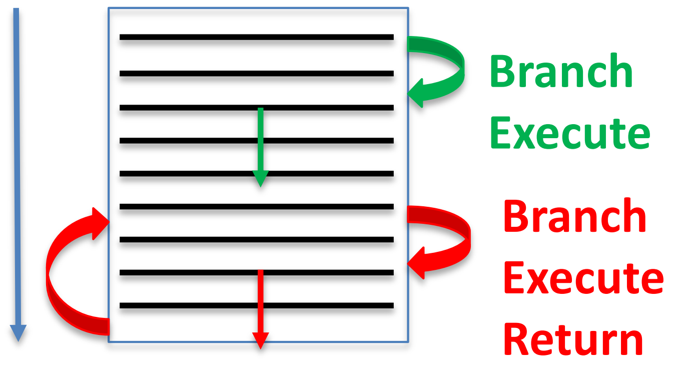
NEG	Negate
ORR	OR
POP	Pop registers
PUSH	Push registers
ROR	Rotate Right
SBC	Subtract with Carry
STMIA	Store Multiple
STR	Store word
STRB	Store byte
STRH	Store halfword
SWI	Software Interrupt
SUB	Subtract
TST	Test bits

2.3.5 Instruction Execution

- **Instruction cycle:**
  - Fetch (decode) and Execute
- **Straight-line sequencing:**
  - Instructions executed in **sequential order**
  - PC incremented by 2/4/8 bytes
  - or 16/32/64-bit word

2.3.5 Instruction Sequencing

- **Change of control flow:**
  - Change Program Counter
  - Two Types:
    - Branch or *Interrupt*



Internal and Expected

External and Unpredictable

2.3.6 Branching

- **Conditional branch:**
- Initialize R2 as a counter; e.g., DO 50 times
  - LOOP: Instruction 1

```
LOOP:   Instruction 1
        Instruction 2
        …
        Subtract          R2, R2, #1  ; change status
        Branch > 0 LOOP               ; check status
```

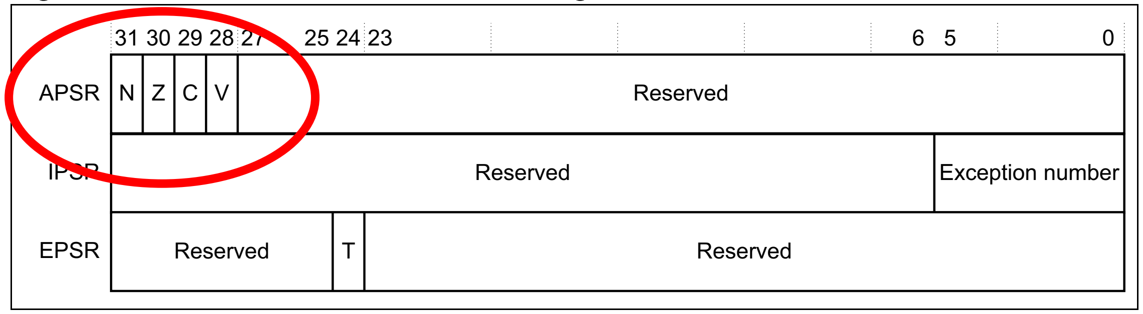
- **Branch target:** LOOP if R2 > 0
- **Condition code** in a **status register (NZCV in APSR)**
  - previous operation results for subsequent conditional use

ARM Conditional Execution (PM39)

- The processor carries out the branch based on the condition code (CC) set by another instruction:

N=negative
Z=zero
C=carry
V=overflow

Figure 3. APSR, IPSR and EPSR bit assignments



- Branch CC placement:
  - Immediately after the instruction that set the condition code, **OR**
  - After any number of in-between instructions that must **not** update the condition code

Branch Placement

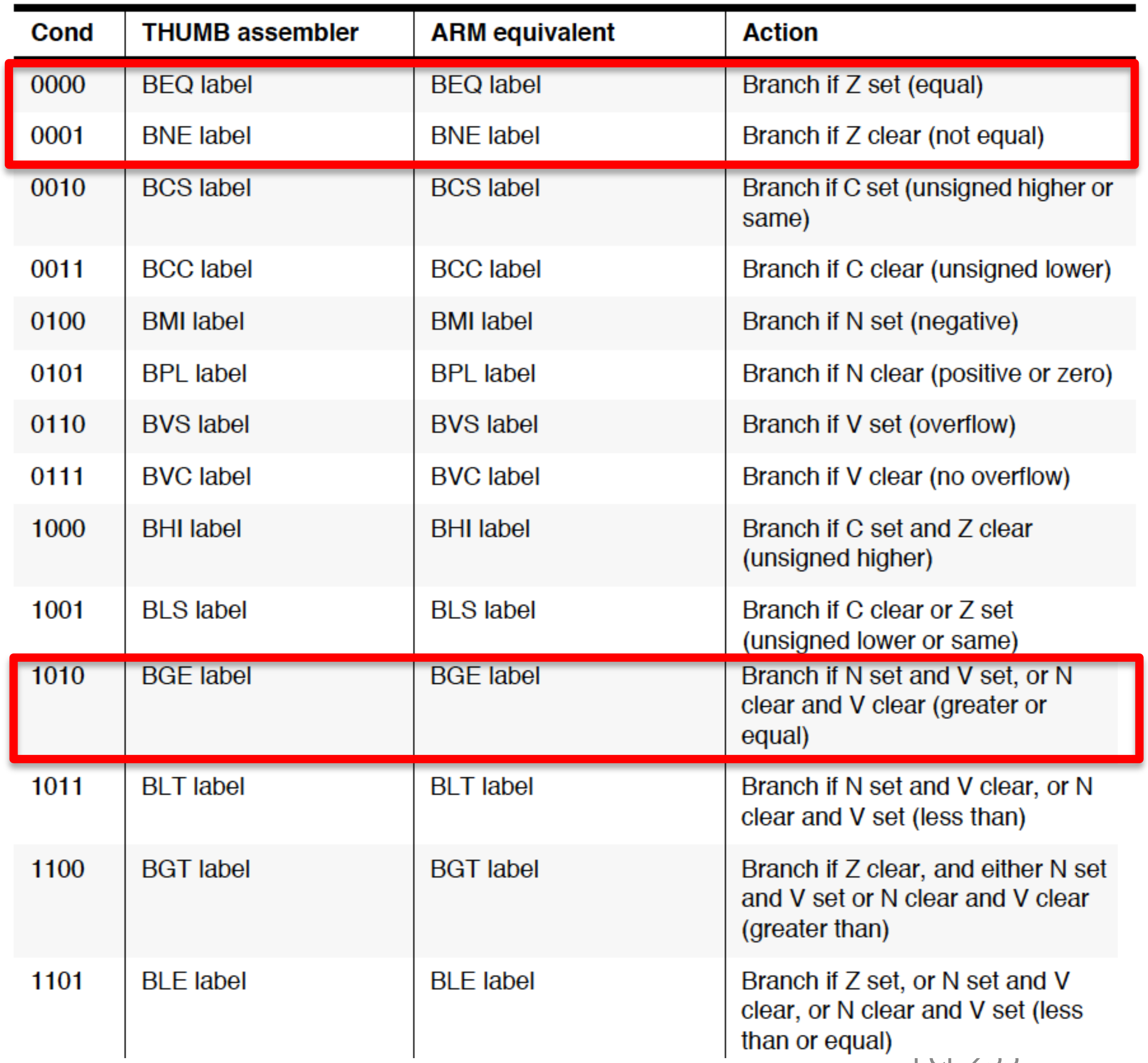
- Initialize R2 as a counter
  - LOOP1:

```
LOOP1:  Instruction 1
        Instruction 2
        …
        Subtract          R2, R2, #1  ; may change status
        Branch > 0 LOOP               ; check status
```

  - LOOP2:

```
LOOP2:  Instruction 1
        Subtract          R2, R2, #1  ; may change status
        …                             ; don't touch Z bit
        Instruction x                 ; don't touch Z bit
        Branch > 0 LOOP               ; check status
```

Thumb Conditional Branch



Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

N=negative
Z=zero
C=carry
V=overflow

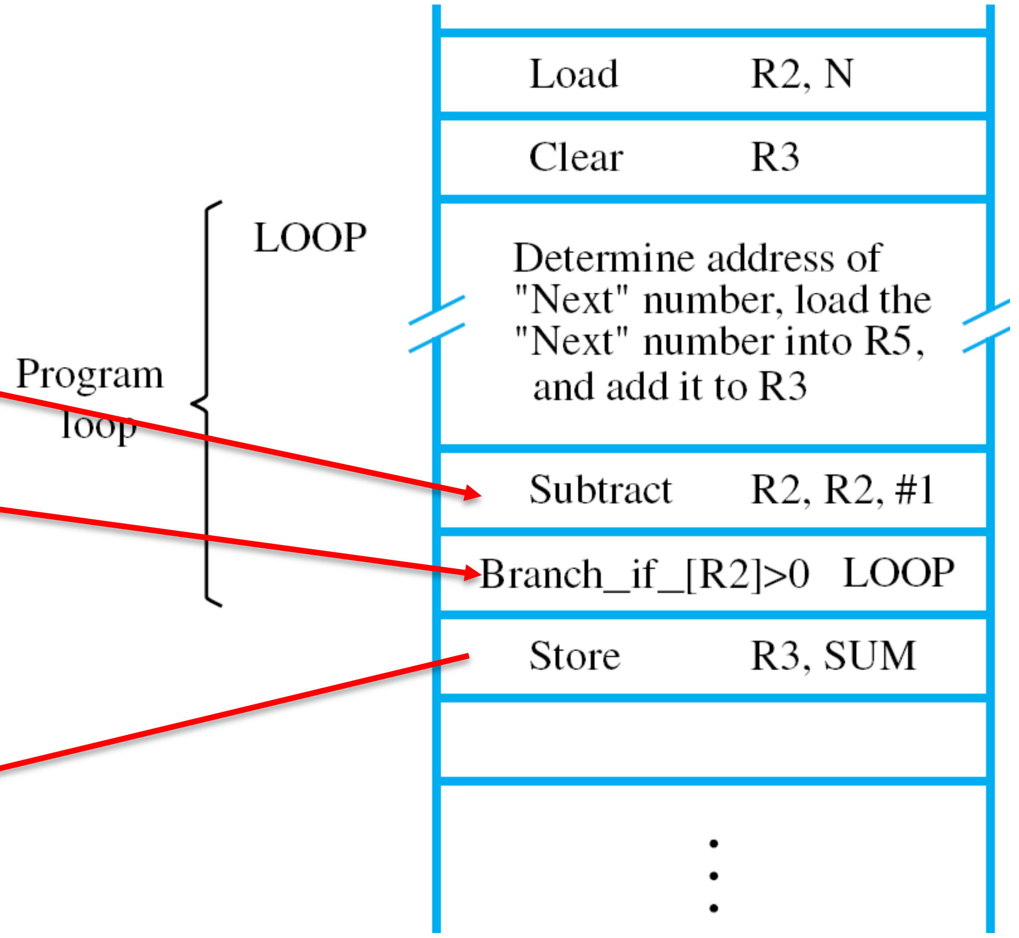
Code = 1110 => undefined

Code = 1111 => SWI

Figure 2.6

```
Initialize R2 as a counter
LOOP:   Instruction 1
        Instruction 2
        …
        Subtract R2, R2, #1
              ; may change status
        Branch > 0     LOOP
              ; check status
```

```
Exit LOOP:
1.   R3 has result of additions
2.   Store in memory location SUM
```


Load R2, N
Clear R3
LOOP
Program loop
Determine address of "Next" number, load the "Next" number into R5, and add it to R3
Subtract R2, R2, #1
Branch_if_[R2]>0 LOOP
Store R3, SUM