Overview:

- Part I
  - Machine Codes
  - Branches and Offsets
  - Subroutine
  - Time Delay
32-Bit ARM Vs. 16/32-Bit THUMB2 Assembly Instructions

**ARM Instruction Formats**

- \( S \) = For data processing instructions, updates condition codes
- \( S \) = For load/store multiple instructions; execution restricted to supervisor mode
- \( P, \ U, \ W \) = distinguish between different types of addressing mode
- \( B \) = Unsign byte \((B==1)\) or word \((B==0)\) access
- \( L \) = For load/store instructions, Load \((L==1)\) or Store \((L==0)\)
- \( L \) = For branch instructions, is return address stored in link register

![ARM Instruction Formats Diagram](image-url)
32-Bit ARM Vs. 16/32-Bit THUMB2 Assembly Instructions

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>Flags</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>EQ</td>
<td>Z==1</td>
</tr>
<tr>
<td>0001</td>
<td>NE</td>
<td>Z==0</td>
</tr>
<tr>
<td>0010</td>
<td>HS/CS</td>
<td>C==1</td>
</tr>
<tr>
<td>0011</td>
<td>LO/CC</td>
<td>C==0</td>
</tr>
<tr>
<td>0100</td>
<td>MI</td>
<td>N==1</td>
</tr>
<tr>
<td>0101</td>
<td>PL</td>
<td>N==0</td>
</tr>
<tr>
<td>0110</td>
<td>VS</td>
<td>V==1</td>
</tr>
<tr>
<td>0111</td>
<td>VC</td>
<td>V==0</td>
</tr>
<tr>
<td>1000</td>
<td>HI</td>
<td>C==1&amp;&amp;Z==0</td>
</tr>
<tr>
<td>1001</td>
<td>LS</td>
<td>C==0</td>
</tr>
<tr>
<td>1010</td>
<td>GE</td>
<td>N==V</td>
</tr>
<tr>
<td>1011</td>
<td>LT</td>
<td>N!=V</td>
</tr>
<tr>
<td>1100</td>
<td>GT</td>
<td>Z==0&amp;&amp;N==V</td>
</tr>
<tr>
<td>1101</td>
<td>LE</td>
<td>Z==1</td>
</tr>
<tr>
<td>1110</td>
<td>AL</td>
<td>always</td>
</tr>
<tr>
<td>1111</td>
<td>NE</td>
<td>never</td>
</tr>
</tbody>
</table>

(u) = unsigned
ARM & THUMB2
Assembly Instructions

32-bit Instruction

16-bit Instruction

Machine Code (16-32 bit)

Assembly Instruction

Address
Conditional Branch – THUMB2 ASSEMBLY

Figure 5-1: THUMB instruction set formats

- Machine Code (16-32 bit)
- Assembly Instruction
- Address
- Memory

Code Example:
```
main:
    r1, #0x15
    mov r1, #0x25
    mov r2, #0x26
    mov r3, #0x27
    mov r4, #0x28
    mov 0x8000256, E7FE
```

Table of THUMB2 Assembly Instructions:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>MOV</td>
<td>Move register</td>
</tr>
<tr>
<td>0001</td>
<td>ADD</td>
<td>Add register</td>
</tr>
<tr>
<td>0010</td>
<td>SUB</td>
<td>Subtract register</td>
</tr>
<tr>
<td>0011</td>
<td>AND</td>
<td>AND register</td>
</tr>
<tr>
<td>0100</td>
<td>OR</td>
<td>OR register</td>
</tr>
<tr>
<td>0101</td>
<td>XOR</td>
<td>XOR register</td>
</tr>
<tr>
<td>0110</td>
<td>STR</td>
<td>Store register</td>
</tr>
<tr>
<td>0111</td>
<td>LDR</td>
<td>Load register</td>
</tr>
<tr>
<td>1000</td>
<td>BAL</td>
<td>Branch and link</td>
</tr>
<tr>
<td>1001</td>
<td>BX</td>
<td>Branch and exchange</td>
</tr>
<tr>
<td>1010</td>
<td>BX</td>
<td>Branch and exchange</td>
</tr>
<tr>
<td>1011</td>
<td>BX</td>
<td>Branch and exchange</td>
</tr>
<tr>
<td>1100</td>
<td>BX</td>
<td>Branch and exchange</td>
</tr>
<tr>
<td>1101</td>
<td>BX</td>
<td>Branch and exchange</td>
</tr>
</tbody>
</table>
Forward Branch –
THUMB2 ASSEMBLY

- Instruction 0x8000244 is an 8-bit branch
- Machine code is E005; Opcode = E0, Offset is 0X05=5
- The next instruction will be:
  - Offset * 2 + Current_Add + 2*2
- Thus
  \[(5*2)_d+0x244+(4)_d=0x244+0xE=0x252\]
- Similarly:
  - Instruction 0x8000256 is an 8-bit branch
  - Machine code is E7FE; Opcode = E7, Offset is FE=-1
  - The next instruction will be:
    - Offset * 2 + Current_Add + 4
  - Thus
    \[(-1*4)_d+0x256+(4)_d=0x256+0x0=0x256 \text{ (same location)}\]
Backward Branch – THUMB2 ASSEMBLY

1101 0001 <1111 1001>
D 1 < F 9 >
Backward Branch – THUMB2 ASSEMBLY

• Instruction 0x8000252 is an 8-bit branch
• Machine code is D1F9; Opcode = D1, Offset is F9=-7
• The next instruction will be:
• Offset * 2 + Current_Add + 2*2
• Thus
  \((-7*2)_d + 0x252 + (4)_d = 0x252 - 0x10 = 0x248\)
• Similarly:
• Instruction 0x8000256 is an 8-bit branch
• Machine code is E7FE; Opcode = E7, Offset is FE=-1
• The next instruction will be:
• Offset * 2 + Current_Add + 4
• Thus
  \((-1*4)_d + 0x256 + (4)_d = 0x256 - 0x0 = 0x256\) (same location)
How Long Does It Take to Run the Code?

- Measure the time at the beginning of the code using the Performance Analyzer.
- Then, step through until the end of the code.
- One **Instruction Cycle** is equivalent to one **Core Clock Cycle** (Clock Period)

![Performance Analyzer](image)
How Long Does It Take to Run the Code?

• Measure the time at the beginning of the code
• Then, step through until the end of the code
• One Instruction Cycle is equivalent to one Core Clock Cycle (Clock Period)

```
23  ;EXAMPLE OF DELAY
24   ; CHANGE TIME (Alt+F7) TO 10 MHZ
25   ; TO SEE THE TIME: VIEW->ANALYSIS WINDOW->LOGIC ANALYZER
26   DELAY   MOV  R0,#255
27   AGAIN   NOP
28   NOP
29   SUBS  R0,R0,#1
30    BNE    AGAIN
31     MOV   PC,LR
```

Total Instruction Cycles (nsec) = \[1+(1+1+3)\times255+1\] \times \text{Core Clock Cycle:100 nsec = 153,200 nsec}
Time Delay in ARM

- Several factors can impact the time it take to execute the code
  - Cote clock cycle: Instruction Cycle = Core clock cycle \cdot N
  - Chip architecture design: Pipelining implementation, instruction size, ALU size, etc.

- In RISC one Instruction cycle is equivalent to on Cluck Cycle:
  - One Instruction cycle is equivalent to on Machine Cycle = Core clock cycle \cdot N

- Generally, MOV, SUB, ADD, NOP etc. take one Instruction Cycle to execute
- Branches take 3 if branch occurs / one Instruction Cycle if no branch occurs
- It take 3 IC to execute an unconditional branch
Subroutines – Branch and Link

• When we call a subroutine

```
AREA CODE, READONLY
.............code........
BL SUBRTN_1
....more code....

SUBRTN_1
.............code........
BX LR ;RETURN TO CALLER

END
```
Example of a Subroutine

LR maintains the address of the caller

Go back to where it was called

Note the subroutine is assembled right after the code:

Code:

```
LDR R1,=RAM_ADDR
AGAIN MOV R0,#0X55
STRB R0,[R1]
BL DELAY
MOV R0,#0XAA
STRB R0,[R1]
B AGAIN

DELAY LDR R3,=5
L1 SUBS R3,R3,#1
BNE L1
BX LR
```

RETURN to the CALLER

we can use MOV PC,LR
Quiz: Write a program to perform \( X = X + 2 \) 100 times, assume \( X = 2 \)

- Draw the flow chart and implement the program
  - Let’s assume \( X \) is in R2
  - The final results is in R1
  - Assume the counter is in R3
- Show that your code works

- **Answer the following questions:**
  - How long did it take to do complete the code?
  - What’s your Xtat Freq?
  - How many instruction cycles are you going through for each loop?
  - How much memory (in BYTE) your code consumes?
  - Using the assembled code, how many instructions are THUMB2?
  - PC Value:
    - Which register contains the PC value?
    - What is the PC value when the code starts?
    - What is the PC value by the time the code is completed?
    - By the time the code is completed, how many WORDs the PC has moved?
  - In which SRAM locations the code is stored?
  - Decode the following – what does it mean?  
    0x0800024C 1E5B SUBS r3, r3, #1