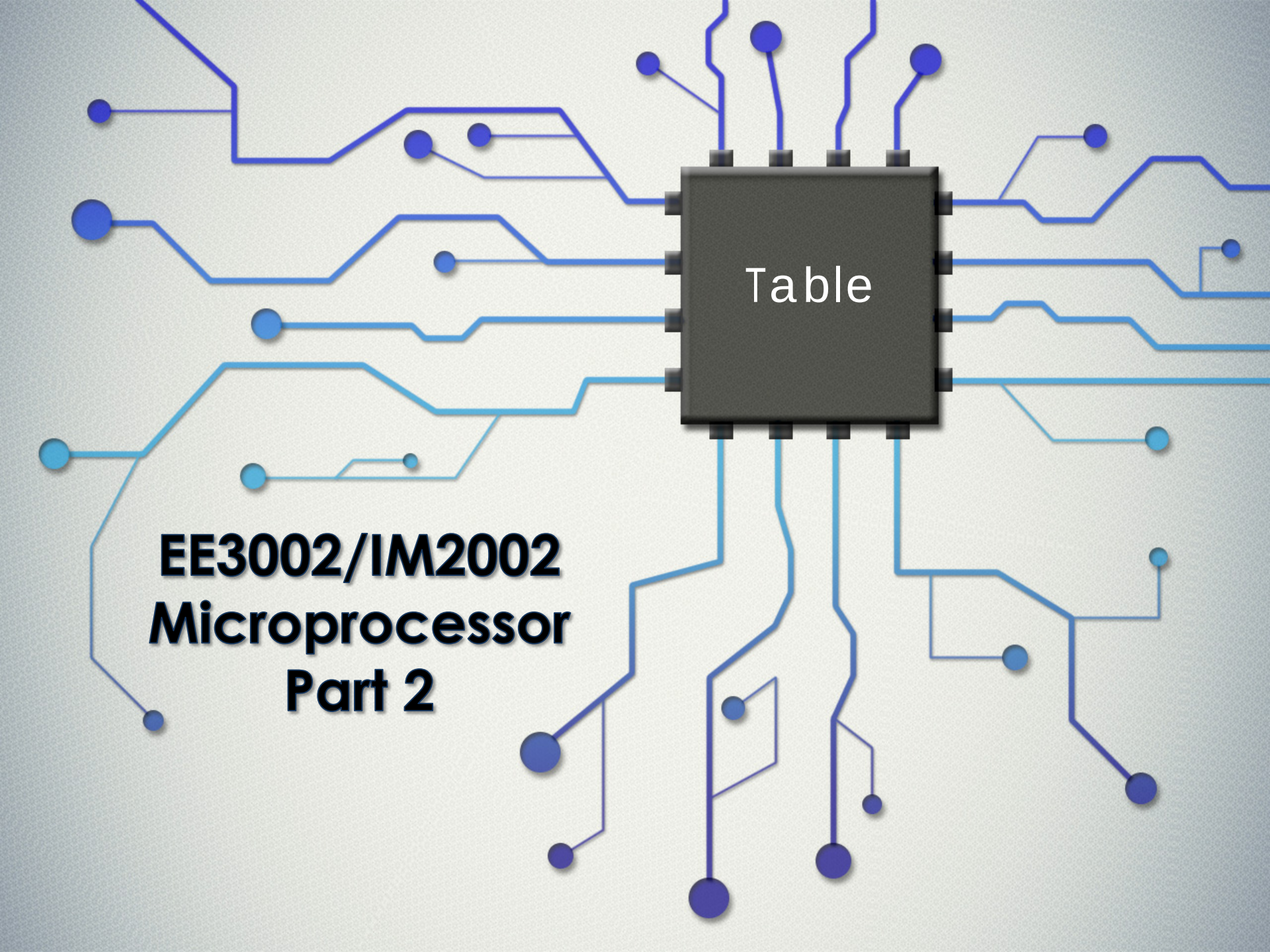


EE3002/IM2002 Microprocessor Part 2

Microprocessors & Microcontrollers

Part 2

- Table
- Stack and Subroutine
- Exception Handling
- THUMB
- Mixing C and Assembly
- Peripherals

Microprocessors & Microcontrollers

TEXTBOOK

- William Hohl, ARM Assembly Language Fundamentals and Techniques, CRC Press, 2nd Edition.

REFERENCES

- Daniel W. Lewis, Fundamentals of Embedded Software with the ARM Cortex-M3, Prentice Hall, 2012.
- Andrew Sloss, Dominic Symes and Chris Wright, ARM System Developer's Guide: Designing and Optimizing System Software, Morgan Kaufmann (Elsevier) 2004.
- David A. Patterson, John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface (ARM edition), 4th edition, Morgan Kaufmann 2011.

Lecturer Information

Name: Dr Chan Chee Keong
Office: S1-B1-B51
Email: eckchan@ntu.edu.sg
Tel: 67905377

Table

Table example 1 – a simple list

- A list of 10 sampled data (eg., weight of young boys)

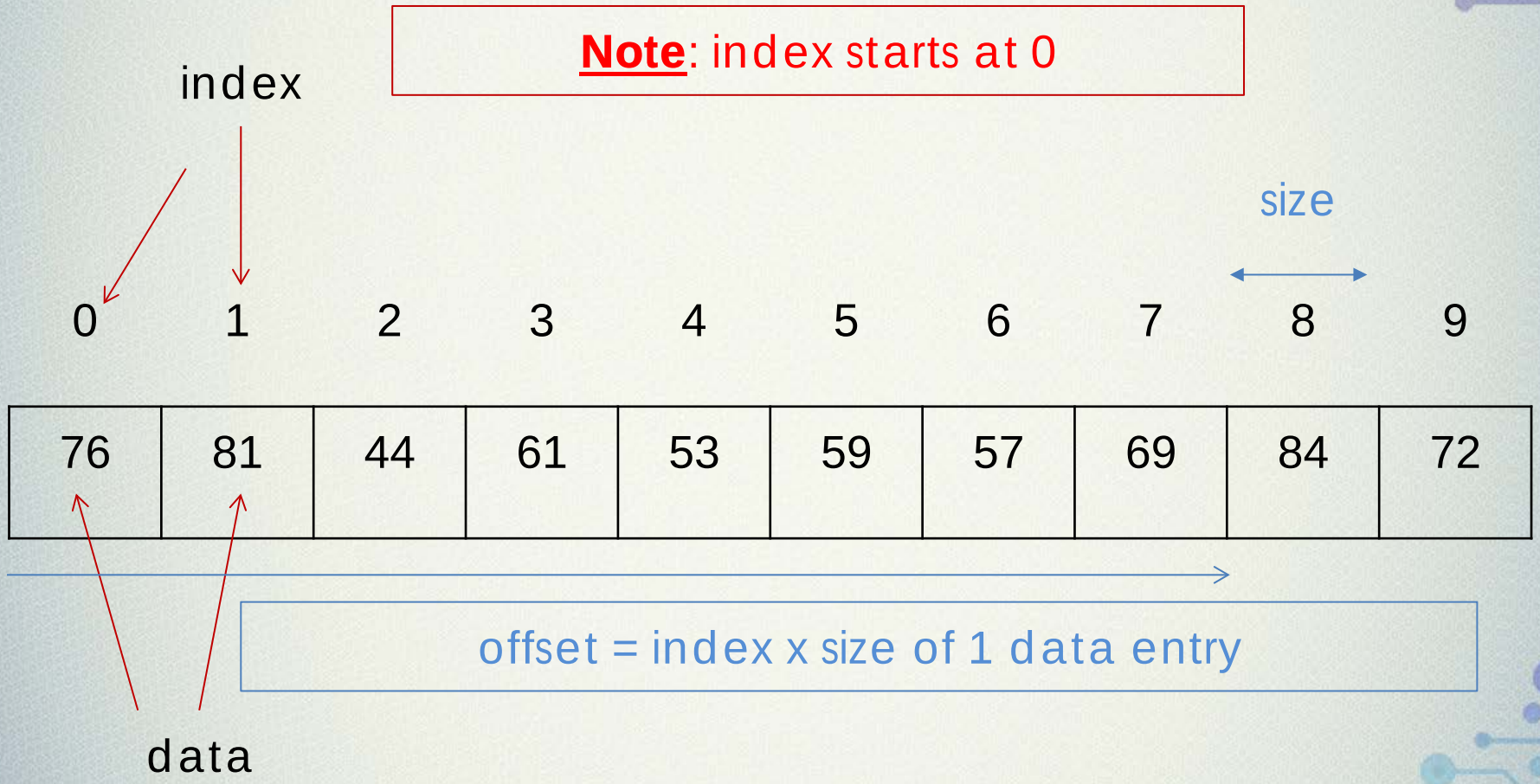
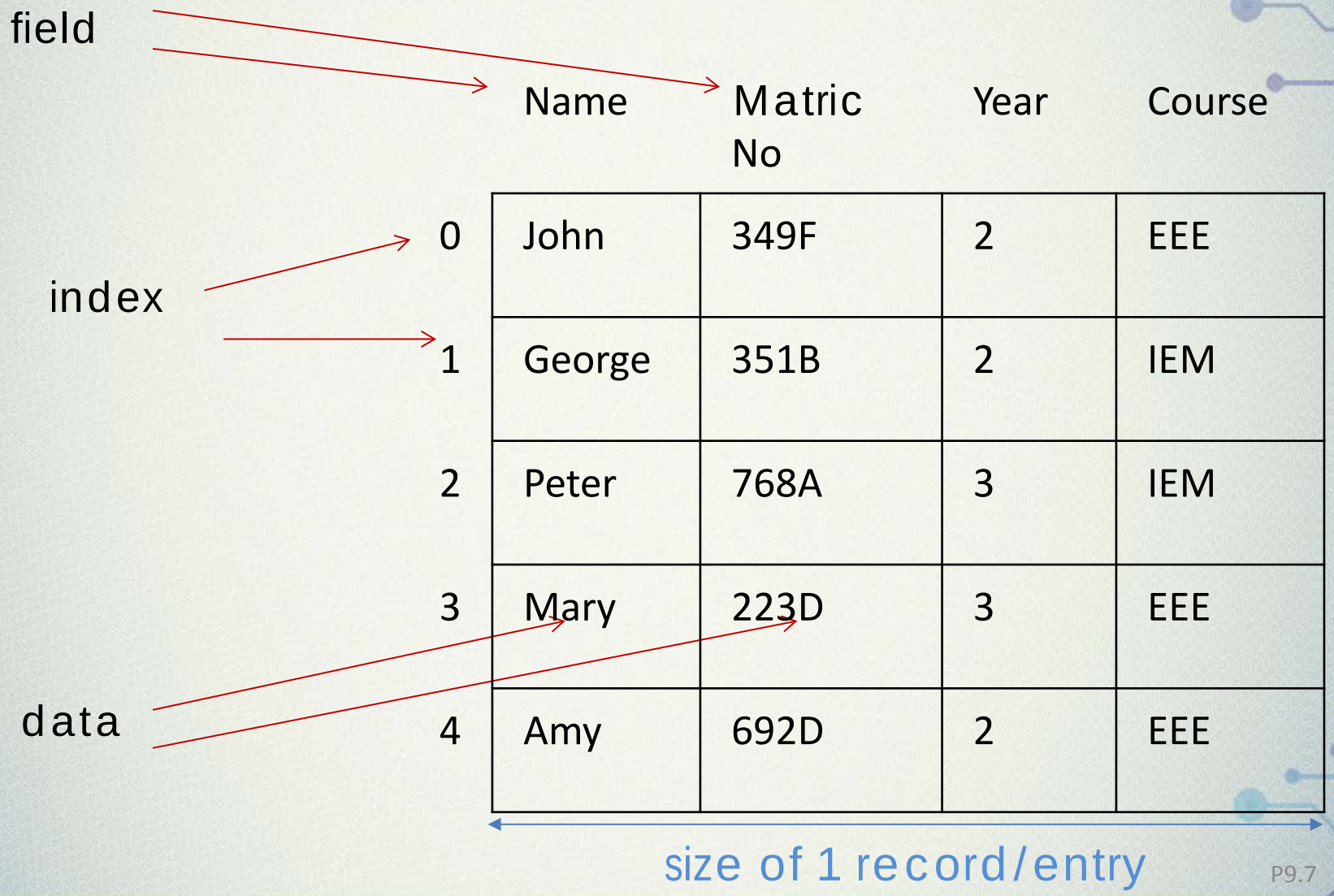


Table example 2 – a more complicated table

- A table contains 5 records of student data



Common tasks on table

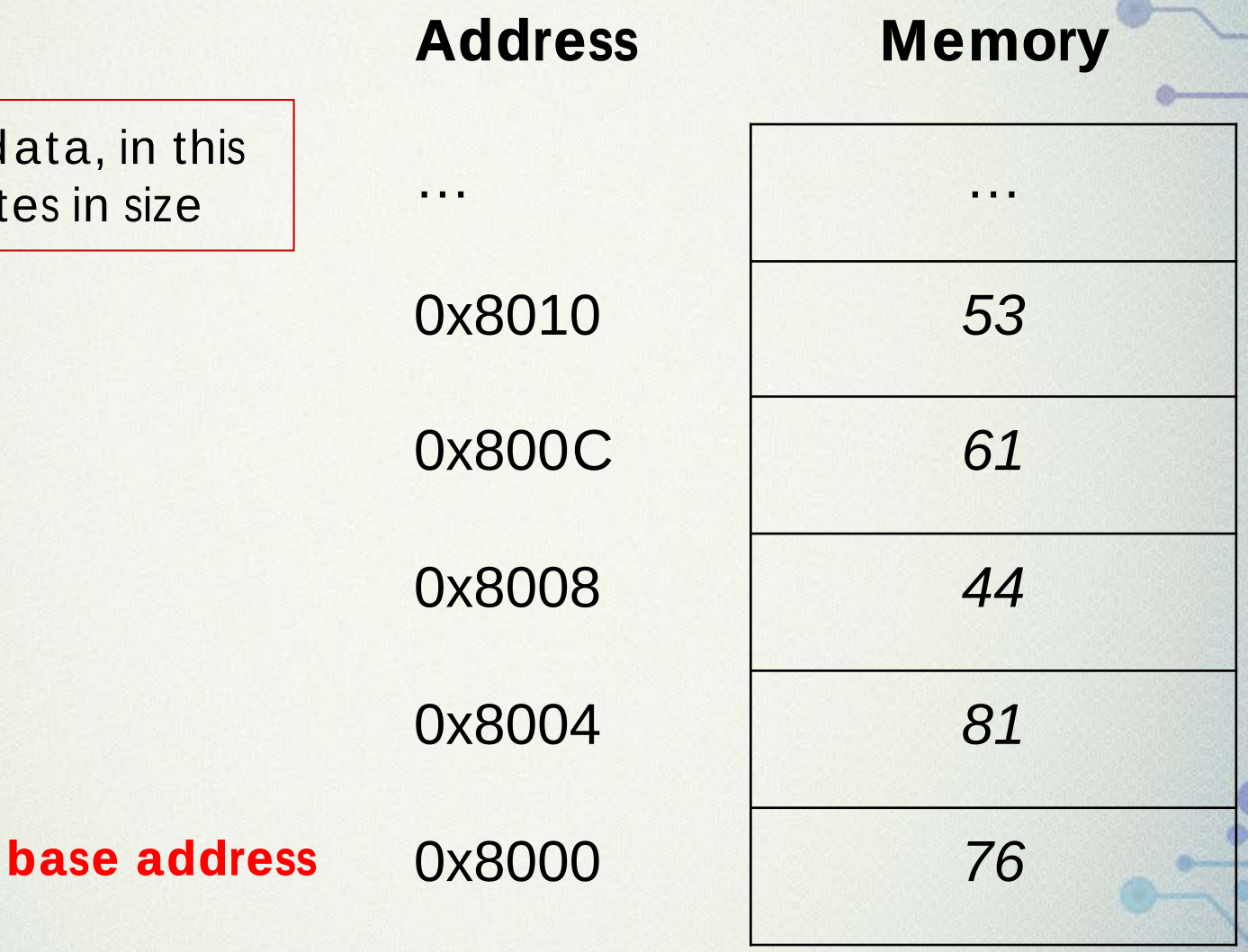
- 2 common tasks are sorting and searching for data in memory from a list of elements , eg., list of sampled data
- Sorting and searching techniques/algorithms could easily fill several books

In this course …

1. Set up table and fill it with data
2. Write efficient algorithms to sort and search/match for certain data in the table

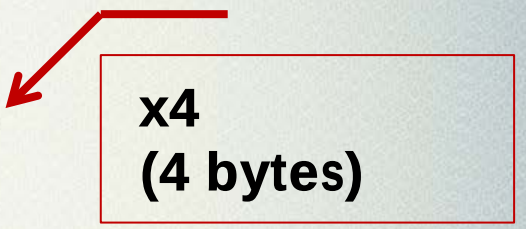
A Simple List in Memory

Note: each data, in this case, is 4 bytes in size

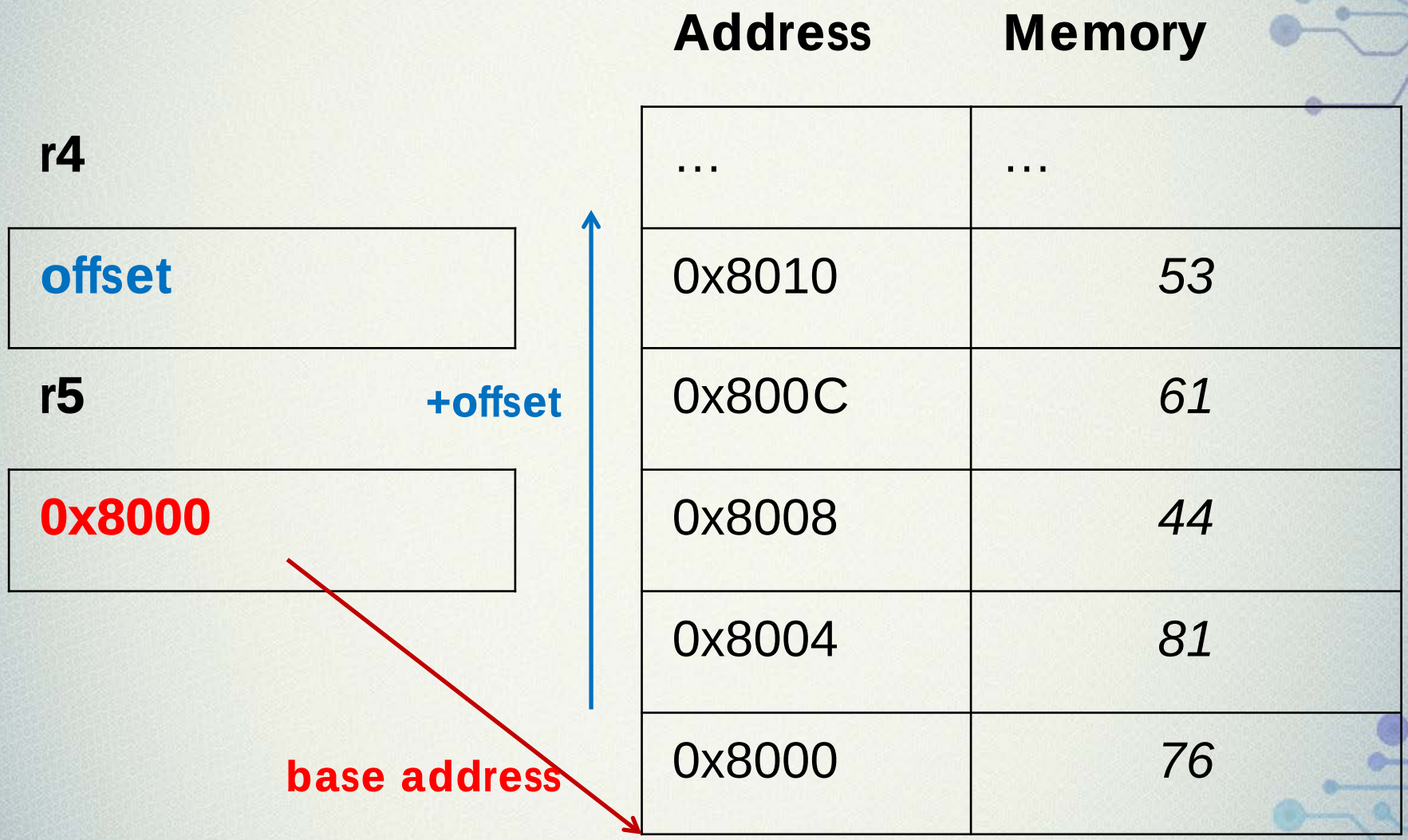


Addressing data of Table

- Use ARM addressing modes: pre-indexed addressing with an offset to address an element
- Example1: **LDR r6, [r5, r4]**
  r4 contains the **offset address**
- Example 2: **LDR r6, [r5, r4, LSL #2]**
  r4 is the **index**
- Note:
  - r5 stores the base (starting) address
  - r6 will be loaded with the addressed data
  - example 2 accounts for the size of the data by multiplying the index by 4 bytes
  - index starts from 0 to (n – 1)

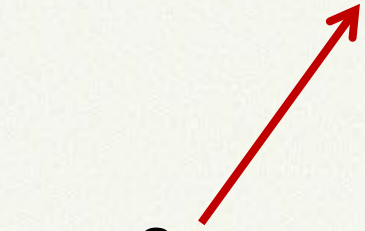


Accessing a Simple List in Memory



Accessing a simple list of data; each data is half word (2 bytes) in size

- Ex1, LDRH r6, [r5, r4] ;r4 - offset
- Ex2, LDRH r6, [r5, r4, LSL #1] ;r4 - index



x2
(2 bytes)

Accessing a simple list of data; each data is 1 byte in size

- Ex1, LDRB r6, [r5, r4] ;r4 – offset or
- Ex2, LDRB r6, [r5, r4] ;r4 - index

Example 9.1 – arithmetic progression

In mathematics, an arithmetic progression (AP) or arithmetic sequence such as

1, 5, 9, 13, 17

has a constant difference between terms. Example, the first term is a_0(=1), the common difference is d(=4), and the number of terms is n(=5). The explicit formula is $a_i = a_{i-1} + d$

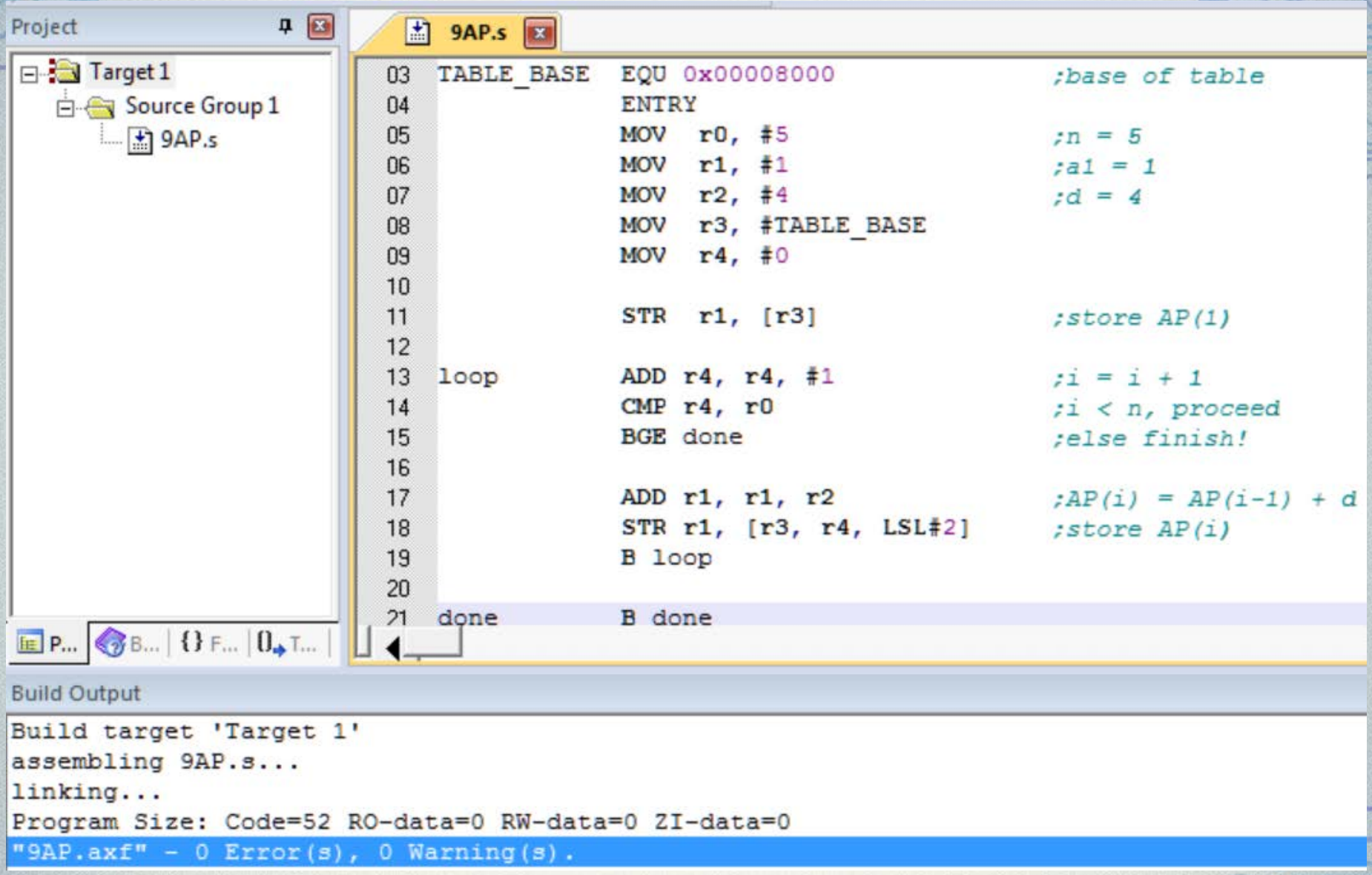
Write an assembly program to create a table of arithmetic progression (AP) in memory. The starting address of the table in memory is 0x8000. Let r0 = n and r1 = a_1 and r2 = d. Assume that the AP are 32-bit data and d is positive.

```
AREA ArithProgression, CODE, READONLY

; Registers used:
; r0 = n = 5
; r1 = AP(i), initial value = a0 = 1
; r2 = d, difference term = 4
; r3 = starting address of table in memory = 0x8000
; r4 = i, loop counter, initially 0
```

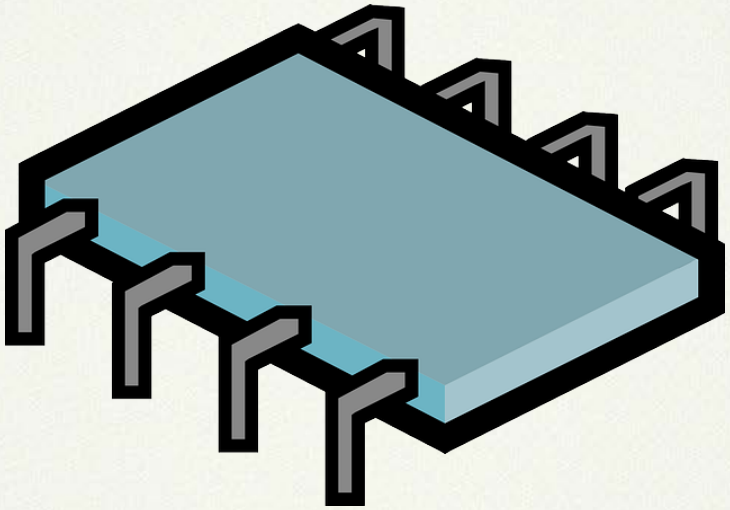
Example 9.1 – AP Codes

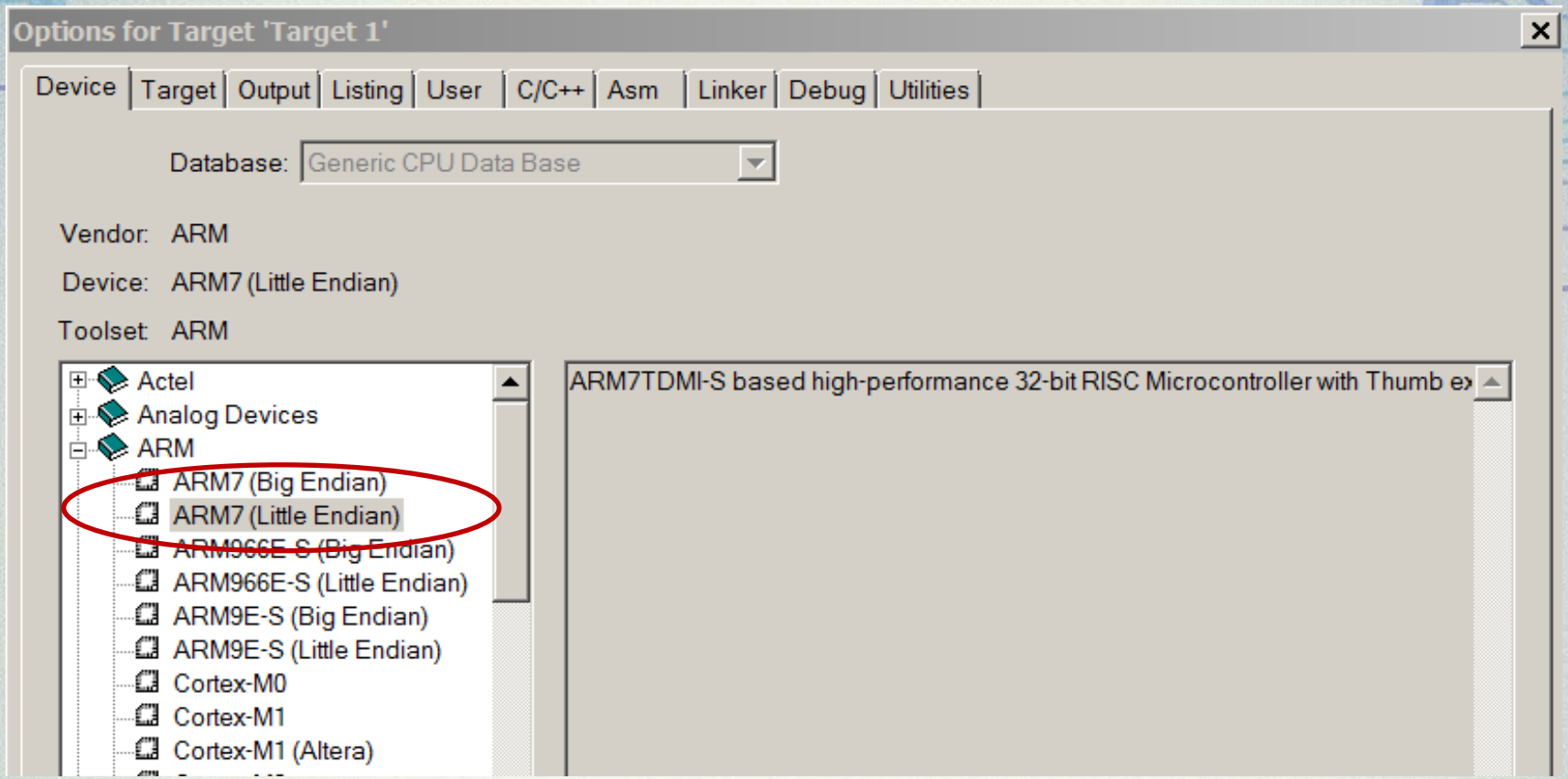
```
TABLE_BASE EQU 0x00008000    ;base of table
    ENTRY
    MOV  r0, #5      ;n = 5
    MOV  r1, #1      ;a0 = 1
    MOV  r2, #4      ;d = 4
    MOV  r3, #TABLE_BASE
    MOV  r4, #0

    STR   r1, [r3]  ;store AP(0)
```

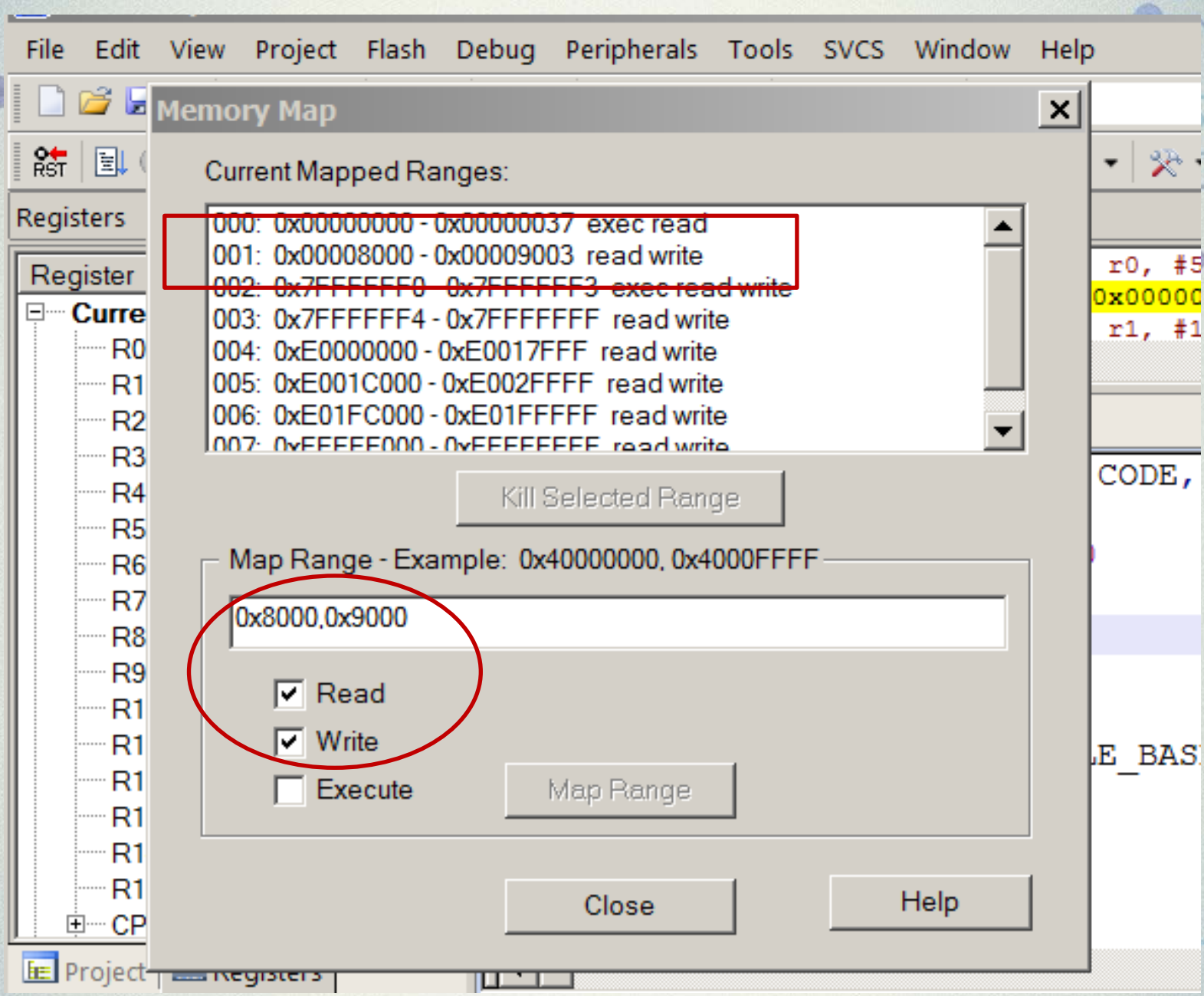
```
loop   ADD r4, r4, #1   ;i = i + 1
       CMP r4, r0       ;i < n, proceed
       BGE done         ;else finish!

       ADD r1, r1, r2   ;AP(i) = AP(i-1) + d
       STR  r1, [r3, r4, LSL #2]  ;store AP(i)
       B loop

done B done
       END
```

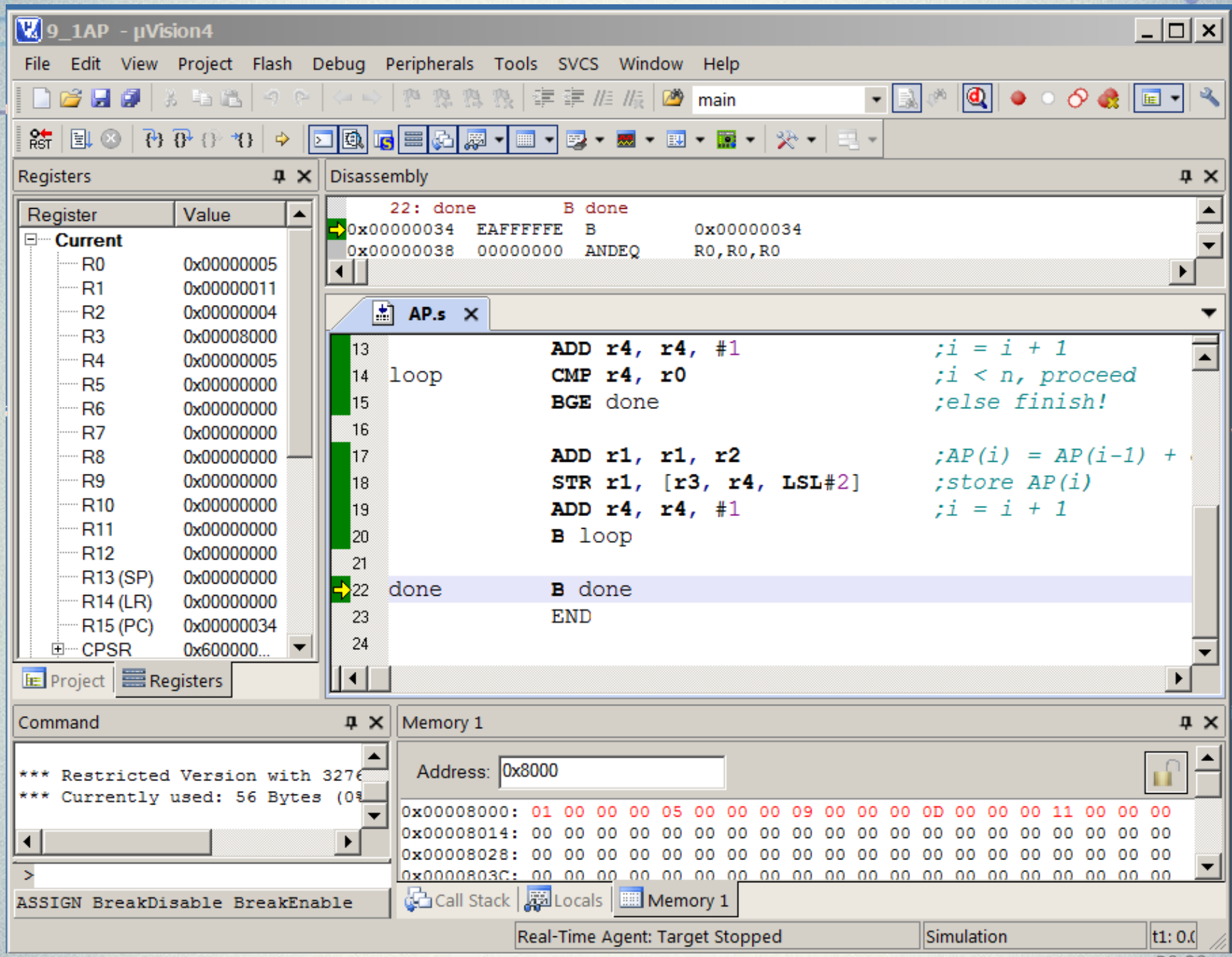

Project
Target 1
Source Group 1
9AP.s
9AP.s
03 TABLE_BASE EQU 0x00008000 ;base of table
04 ENTRY
05 MOV r0, #5 ;n = 5
06 MOV r1, #1 ;a1 = 1
07 MOV r2, #4 ;d = 4
08 MOV r3, #TABLE_BASE
09 MOV r4, #0
10
11 STR r1, [r3] ;store AP(1)
12
13 loop ADD r4, r4, #1 ;i = i + 1
14 CMP r4, r0 ;i < n, proceed
15 BGE done ;else finish!
16
17 ADD r1, r1, r2 ;AP(i) = AP(i-1) + d
18 STR r1, [r3, r4, LSL#2] ;store AP(i)
19 B loop
20
21 done B done
Build Output
Build target 'Target 1'
assembling 9AP.s...
linking...
Program Size: Code=52 RO-data=0 RW-data=0 ZI-data=0
"9AP.axf" - 0 Error(s), 0 Warning(s).

Demo 9.1 – Generate AP




Options for Target 'Target 1'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Database:
Generic CPU Data Base
Vendor: ARM
Device: ARM7 (Little Endian)
Toolset: ARM
Actel
Analog Devices
ARM
ARM7 (Big Endian)
ARM7 (Little Endian)
ARM966E-S (Big Endian)
ARM966E-S (Little Endian)
ARM9E-S (Big Endian)
ARM9E-S (Little Endian)
Cortex-M0
Cortex-M1
Cortex-M1 (Altera)
ARM7TDMI-S based high-performance 32-bit RISC Microcontroller with Thumb ex


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Memory Map
Current Mapped Ranges:
000: 0x00000000 - 0x00000037 exec read
001: 0x00008000 - 0x00009003 read write
002: 0x7FFFFFF0 - 0x7FFFFFF3 exec read write
003: 0x7FFFFFF4 - 0x7FFFFFFF read write
004: 0xE0000000 - 0xE0017FFF read write
005: 0xE001C000 - 0xE002FFFF read write
006: 0xE01FC000 - 0xE01FFFFF read write
Kill Selected Range
Map Range - Example: 0x40000000, 0x4000FFFF
0x8000,0x9000
Read
Write
Execute
Map Range
Close
Help
Registers
Register
Project
CODE,


9_1AP - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
main
Registers
Register Value
Current
R0 0x00000005
R1 0x00000011
R2 0x00000004
R3 0x00008000
R4 0x00000005
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000034
CPSR 0x600000...
Project Registers
Disassembly
22: done B done
0x00000034 EAFFFFFE B 0x00000034
0x00000038 00000000 ANDEQ R0,R0,R0
AP.s
13 ADD r4, r4, #1 ;i = i + 1
14 loop CMP r4, r0 ;i < n, proceed
15 BGE done ;else finish!
16
17 ADD r1, r1, r2 ;AP(i) = AP(i-1) +
18 STR r1, [r3, r4, LSL#2] ;store AP(i)
19 ADD r4, r4, #1 ;i = i + 1
20 B loop
21
22 done B done
23 END
24
Command
*** Restricted Version with 3276
*** Currently used: 56 Bytes (0%
ASSIGN BreakDisable BreakEnable
Memory 1
Address: 0x8000
0x00008000: 01 00 00 00 05 00 00 00 09 00 00 00 0D 00 00 00 11 00 00 00
0x00008014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x00008028: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Call Stack Locals Memory 1
Real-Time Agent: Target Stopped
Simulation
t1: 0.0

Jump Table

- Instead of holding data, these tables hold addresses of subroutines.
- Useful for replacing a series of comparisons and branches.
- For example, a controller may receive an input from a keypad, and based on the number that was pressed, jump to a particular subroutine to configure registers, display data, or initialize variables.
- Much like the switch statement in C.

Example 9.2 Jump Table

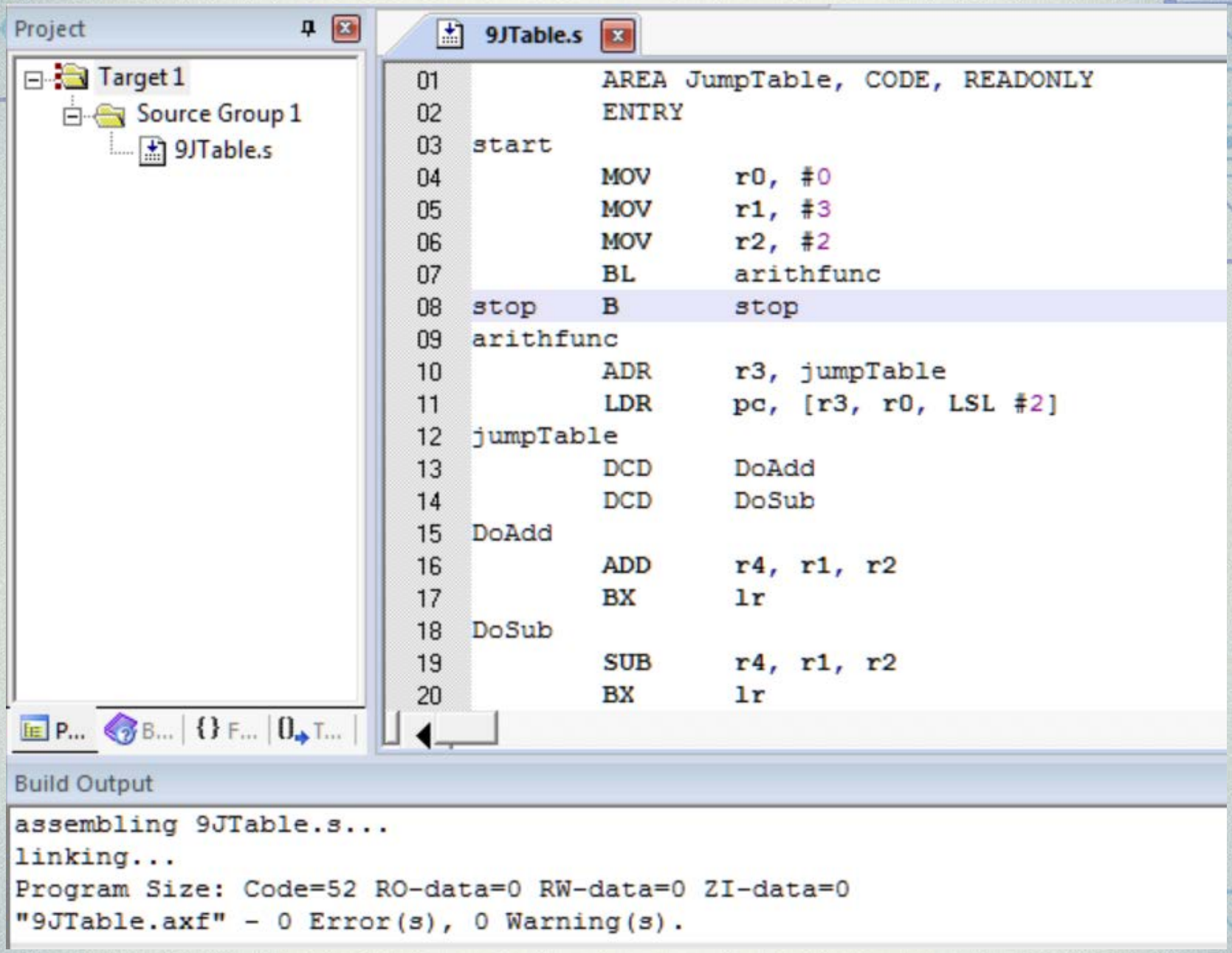
- function **arithfunc** takes three arguments
- The first argument determines the operation to be performed on the other two arguments
- If the first argument is a “0”, the returned result is the sum of the second and third arguments
- If the first argument is a “1”, the returned result is the difference of the second and third arguments

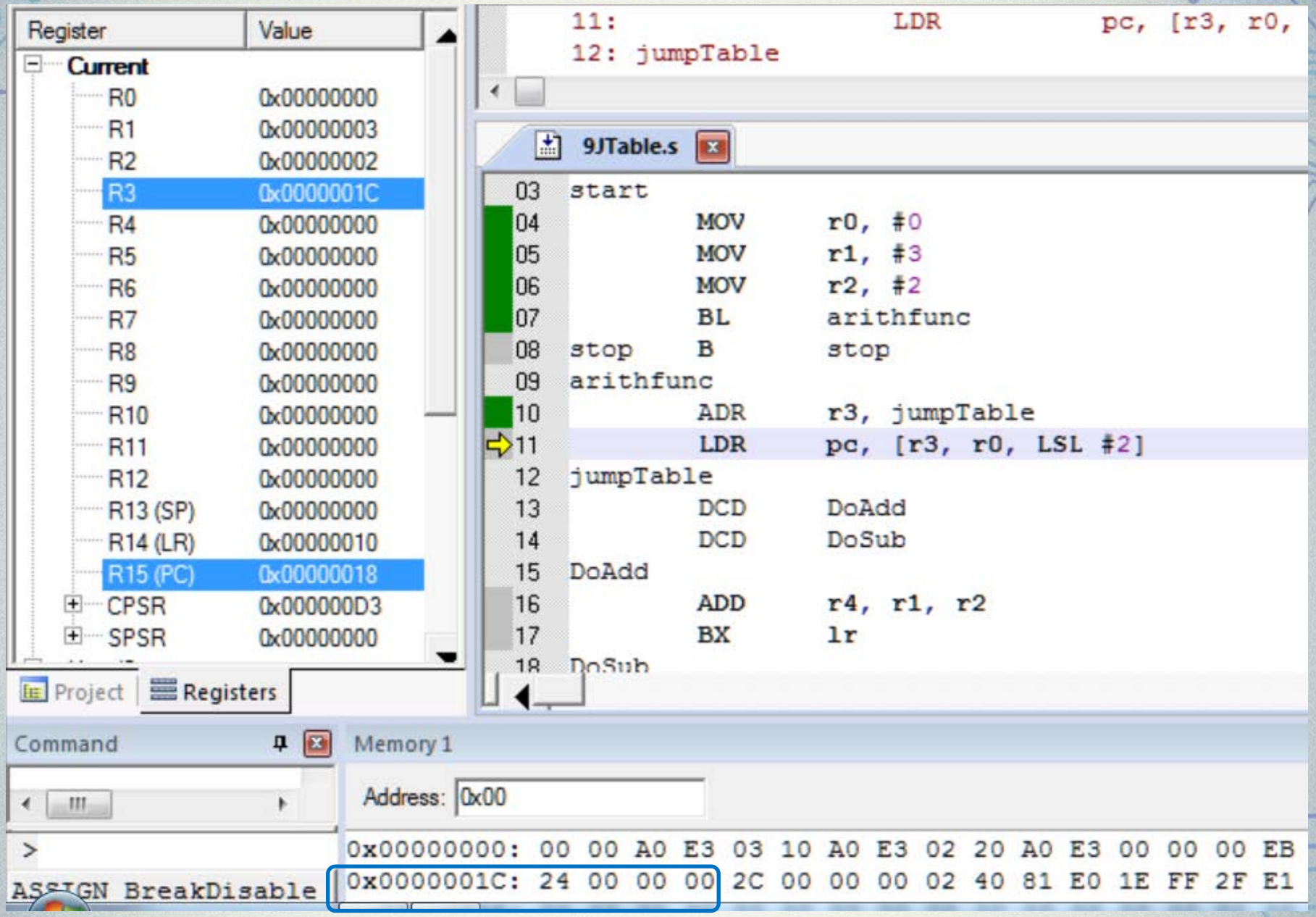
Example 9.2 Jump Table Codes

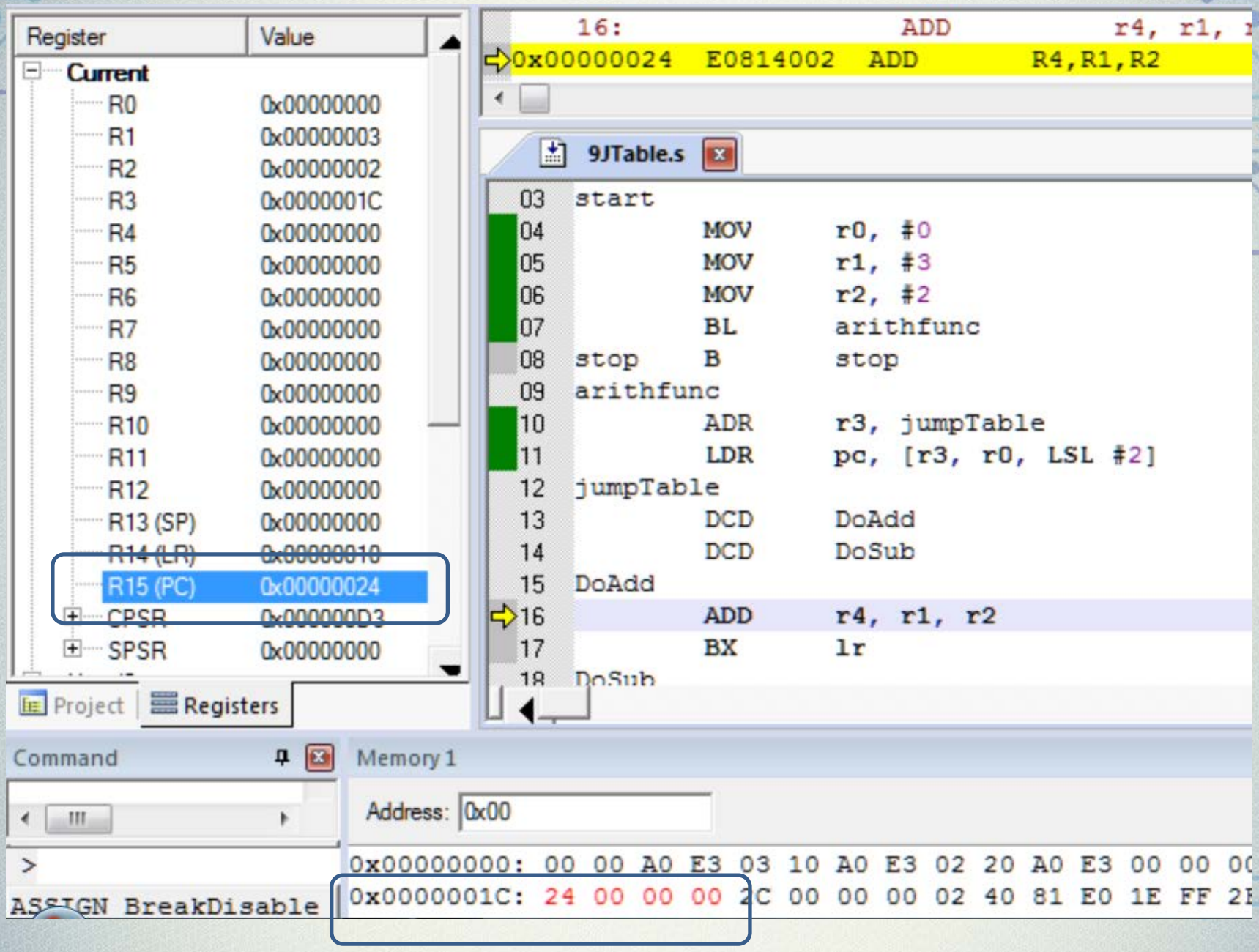
```
    AREA Jump, CODE, READONLY
    ENTRY
Start
    MOV  r0, #0 ; option = 0 implies do addition
    MOV  r1, #3 ;2nd argument
    MOV  r2, #2 ;3rd argument
    BL  arithfunc; call the function
stop    B  stop
```

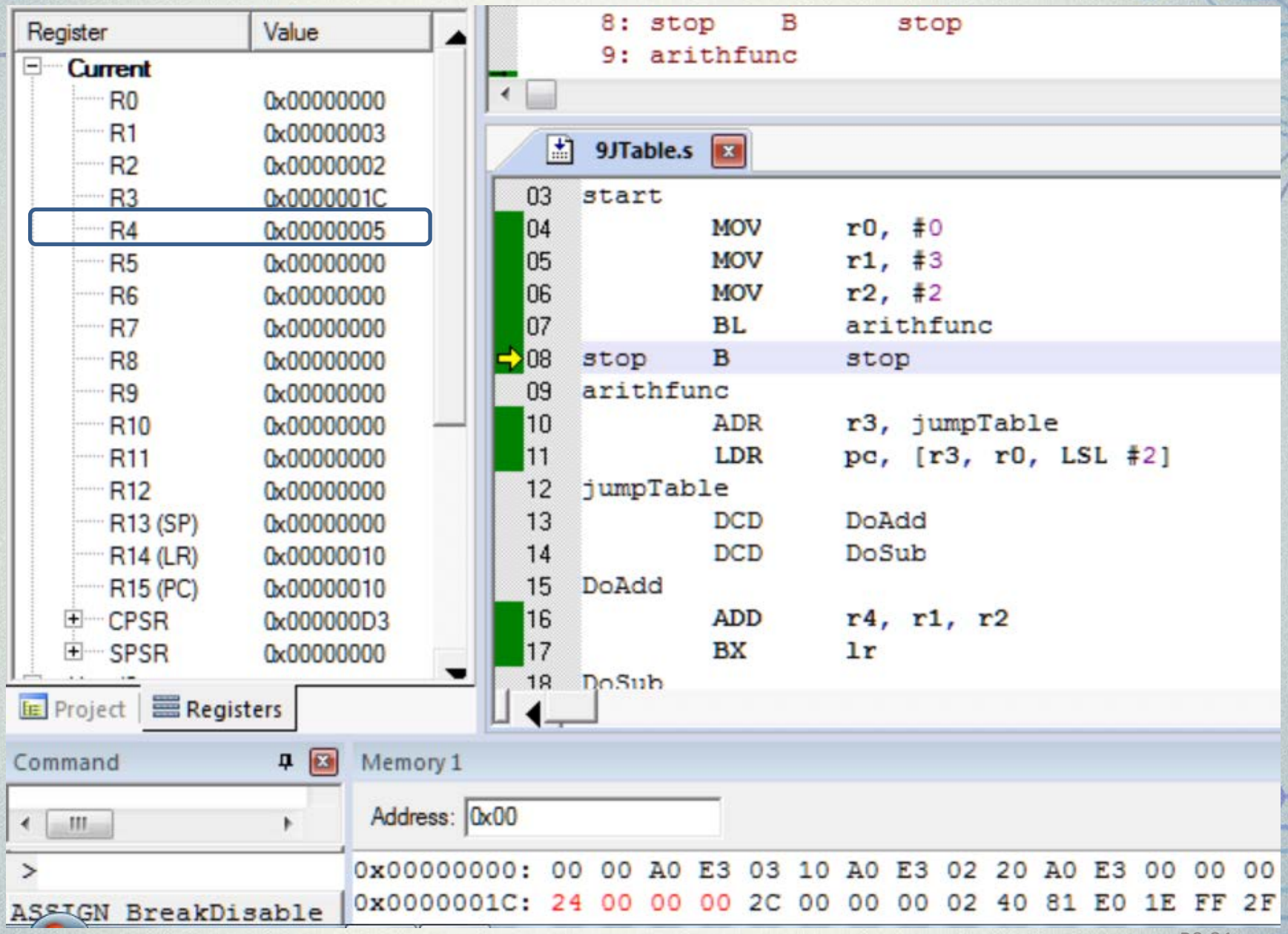
Example 9.2 Jump Table Codes - cont

```
arithfunc       ; label the function
    ADR  r3, jumpTable  ; load address of jump table
    LDR  pc, [r3, r0, LSL #2]  ; jump to the appropriate routine
jumpTable
    DCD  DoAdd ;r0=0(add), contains address of DoAdd
    DCD  DoSub  ;r0=1(sub), contains address of DoSub
DoAdd
    ADD  r4, r1, r2; operation DoAdd, when r0 = 0
    BX    pc, lr     ; return to calling program
DoSub
    SUB  r4, r1, r2  ; operation DoSub, when r0 = 1
    BX    lr      ; return to calling program
    END           ; Mark the end of this file
```


Project
Target 1
Source Group 1
9JTable.s
P...
B...
F...
T...
9JTable.s
01 AREA JumpTable, CODE, READONLY
02 ENTRY
03 start
04 MOV r0, #0
05 MOV r1, #3
06 MOV r2, #2
07 BL arithfunc
08 stop B stop
09 arithfunc
10 ADR r3, jumpTable
11 LDR pc, [r3, r0, LSL #2]
12 jumpTable
13 DCD DoAdd
14 DCD DoSub
15 DoAdd
16 ADD r4, r1, r2
17 BX lr
18 DoSub
19 SUB r4, r1, r2
20 BX lr
Build Output
assembling 9JTable.s...
linking...
Program Size: Code=52 RO-data=0 RW-data=0 ZI-data=0
"9JTable.axf" - 0 Error(s), 0 Warning(s).


Register
Value
Current
R0 0x00000000
R1 0x00000003
R2 0x00000002
R3 0x0000001C
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000010
R15 (PC) 0x00000018
CPSR 0x000000D3
SPSR 0x00000000
Project
Registers
11: LDR pc, [r3, r0,
12: jumpTable
9JTable.s
03 start
04 MOV r0, #0
05 MOV r1, #3
06 MOV r2, #2
07 BL arithfunc
08 stop B stop
09 arithfunc
10 ADR r3, jumpTable
11 LDR pc, [r3, r0, LSL #2]
12 jumpTable
13 DCD DoAdd
14 DCD DoSub
15 DoAdd
16 ADD r4, r1, r2
17 BX lr
18 DoSub
Command
Memory 1
Address: 0x00
>
ASSIGN BreakDisable
0x00000000: 00 00 A0 E3 03 10 A0 E3 02 20 A0 E3 00 00 00 EB
0x0000001C: 24 00 00 00 2C 00 00 00 02 40 81 E0 1E FF 2F E1


Register
Value
Current
R0 0x00000000
R1 0x00000003
R2 0x00000002
R3 0x0000001C
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000010
R15 (PC) 0x00000024
CPSR 0x000000D3
SPSR 0x00000000
Project
Registers
16: ADD r4, r1, r
0x00000024 E0814002 ADD R4,R1,R2
9JTable.s
03 start
04 MOV r0, #0
05 MOV r1, #3
06 MOV r2, #2
07 BL arithfunc
08 stop B stop
09 arithfunc
10 ADR r3, jumpTable
11 LDR pc, [r3, r0, LSL #2]
12 jumpTable
13 DCD DoAdd
14 DCD DoSub
15 DoAdd
16 ADD r4, r1, r2
17 BX lr
18 DoSub
Command
>
ASSIGN BreakDisable
Memory 1
Address: 0x00
0x00000000: 00 00 A0 E3 03 10 A0 E3 02 20 A0 E3 00 00 0
0x0000001C: 24 00 00 00 2C 00 00 00 02 40 81 E0 1E FF 2


Register
Value
Current
R0 0x00000000
R1 0x00000003
R2 0x00000002
R3 0x0000001C
R4 0x00000005
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000010
R15 (PC) 0x00000010
CPSR 0x000000D3
SPSR 0x00000000
Project
Registers
8: stop B stop
9: arithfunc
9JTable.s
03 start
04 MOV r0, #0
05 MOV r1, #3
06 MOV r2, #2
07 BL arithfunc
08 stop B stop
09 arithfunc
10 ADR r3, jumpTable
11 LDR pc, [r3, r0, LSL #2]
12 jumpTable
13 DCD DoAdd
14 DCD DoSub
15 DoAdd
16 ADD r4, r1, r2
17 BX lr
18 DoSub
Command
ASSIGN BreakDisable
Memory 1
Address: 0x00
0x00000000: 00 00 A0 E3 03 10 A0 E3 02 20 A0 E3 00 00 00
0x0000001C: 24 00 00 00 2C 00 00 00 02 40 81 E0 1E FF 2F

Exercise: Change r0 = 1 and observe the change in operation!

Bubble Sort

- In order to search, we need to sort the table (list) first
- A bubble sort is a simple way to sort entries
- Algorithm
  - Compare 2 entries in a list (entry [j] and entry[j+1])
  - If entry[j] is larger, then swap the entries
  - Repeat this until the last 2 entries are compared
  - The largest element will now be the last entry
  - This is repeated until the smallest element is swapped or bubbled to be the first entry

Bubble Sort – Pseudo code

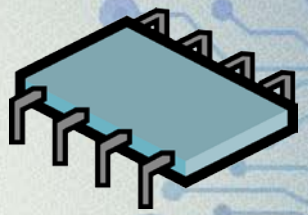
```
/*n is the number of entries*/
i=0;
while (i < n) {          //outer loop, i is loop counter
   j = 0;
   while (j < (n-1)) {  //inner loop, j is loop counter
      if (entry[j] > entry[j+1]) {
        temp = entry[j];      //swap
        entry[j] = entry[j+1];         //if
        entry[j+1] = temp;       //greater
      }
      j = j+1;
   }
  i = i+1;
}
```

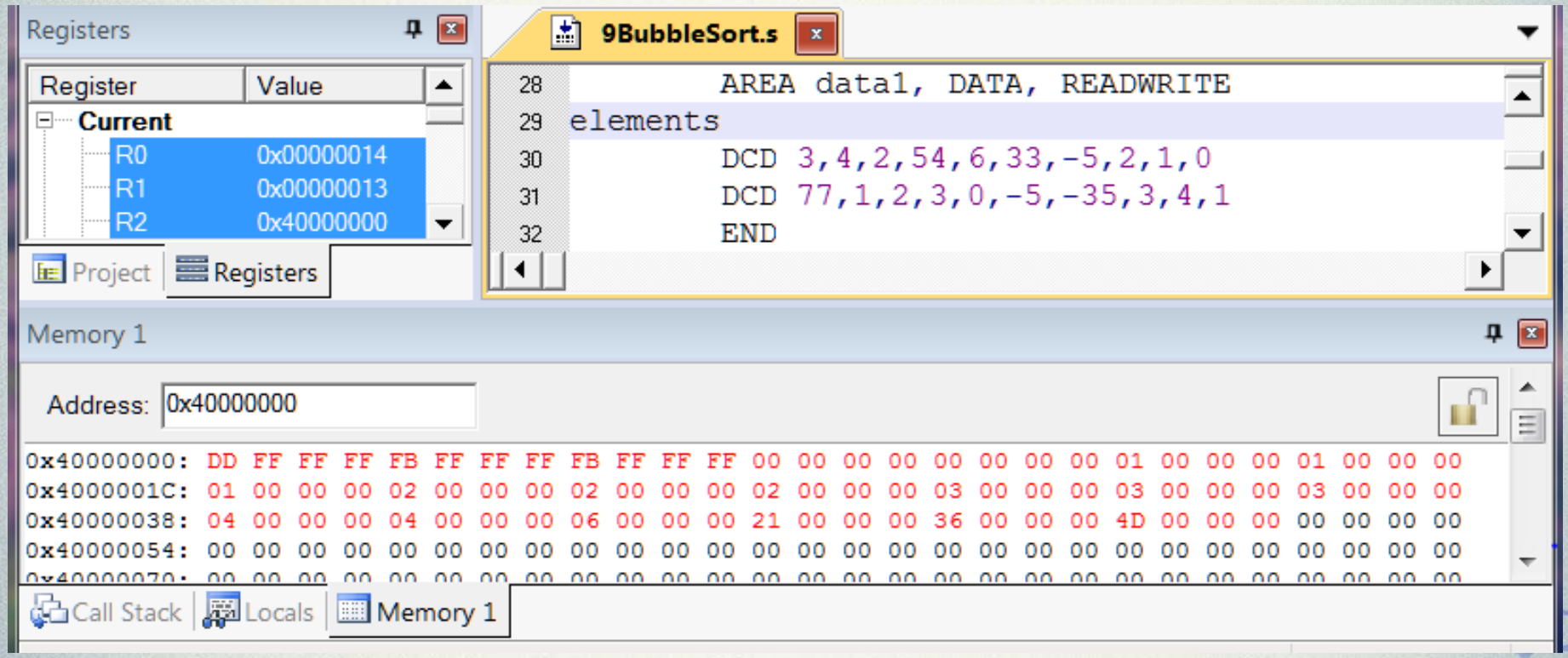
Example 9.3 Bubble Sort – ARM codes

```
        AREA BubbleSort, CODE, READONLY
num EQU  20
        ENTRY
        MOV  r0, #num    ;n
        SUB   r1, r0, #1 ;n-1
        LDR   r2, =elements  ;base address of table
        MOV  r3, #0    ;loop counter i
```

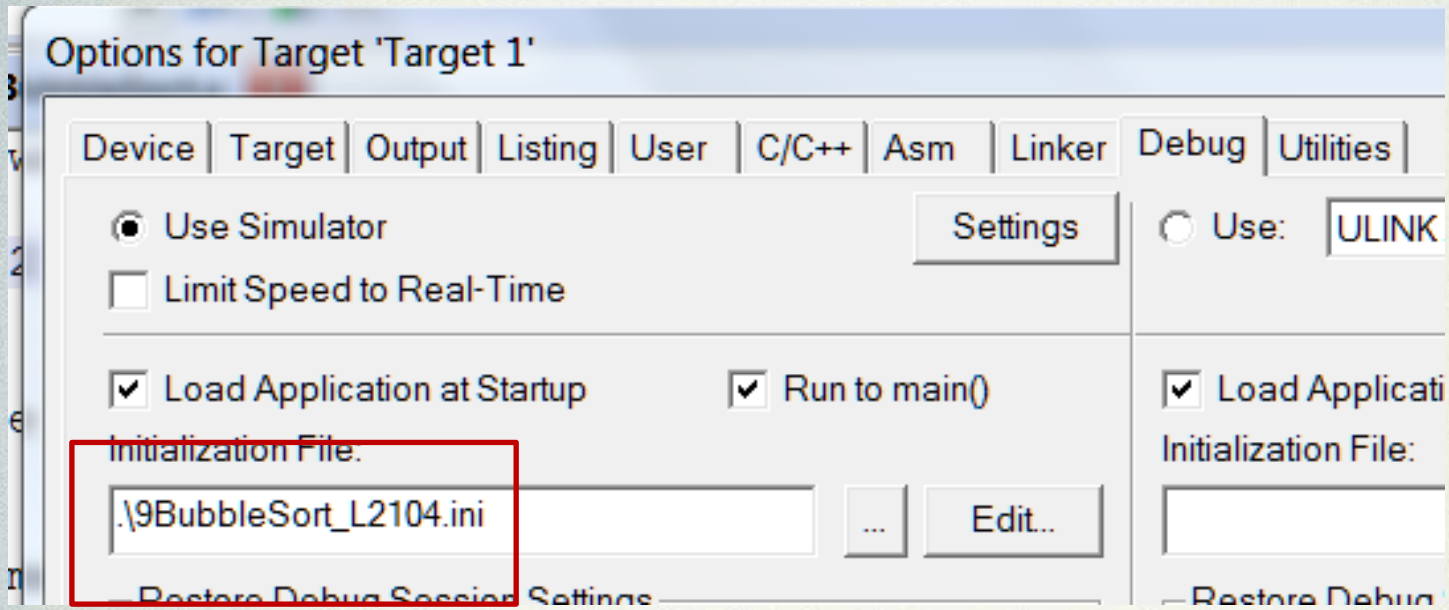
```
loop1   CMP  r3, r0  ;i < n
        BGE   done     ;done
        MOV r4, #0     ;loop counter j
loop2   CMP  r4, r1;j < (n-1)
        BGE   end2
        ADD  r5, r4, #1;j+1
        LDR   r6, [r2, r4, LSL #2];load entry[j]
        LDR   r7, [r2, r5, LSL #2];load entry[j+1]
        CMP  r6, r7
        BLE   Noswap
        STR   r6, [r2, r5, LSL #2] ;swap if necessary
        STR   r7, [r2, r4, LSL #2]
Noswap
        ADD r4, r4, #1  ;inc j
        B      loop2
end2    ADD r3, r3, #1 ;inc i
        B      loop1

done    B       done
```

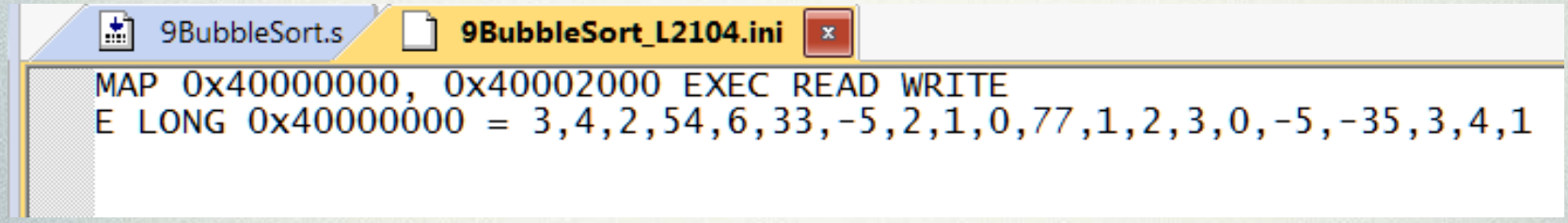
```
      AREA  data1, DATA,READWRITE
elements
      DCD   3, 4, 2, 54, 6, 33, -5, 2, 1, 0
      DCD   77, 1, 2, 3, 0, -5, -35, 3, 4, 1
END
```





Example 9.3 Bubble Sort – initial file





Search for data in a table

- It is common to search a table for information, using a key
- Once the key matches an item in a table, the address of the matched item is noted and further processing can be done
- The simplest way is to search the table from the beginning to the end of the table → **sequential search**
- Another more efficient way is to use **binary search,** which requires the table to be sorted, either in ascending or descending order

Sequential Search – Pseudo code

```
found = 0;
n =  number of items in table;
index = 0;
while   ((index < n) && (found == 0)) {
     if (key == table[index])
          found = address of matched item;
     else index = index + 1;
}
```

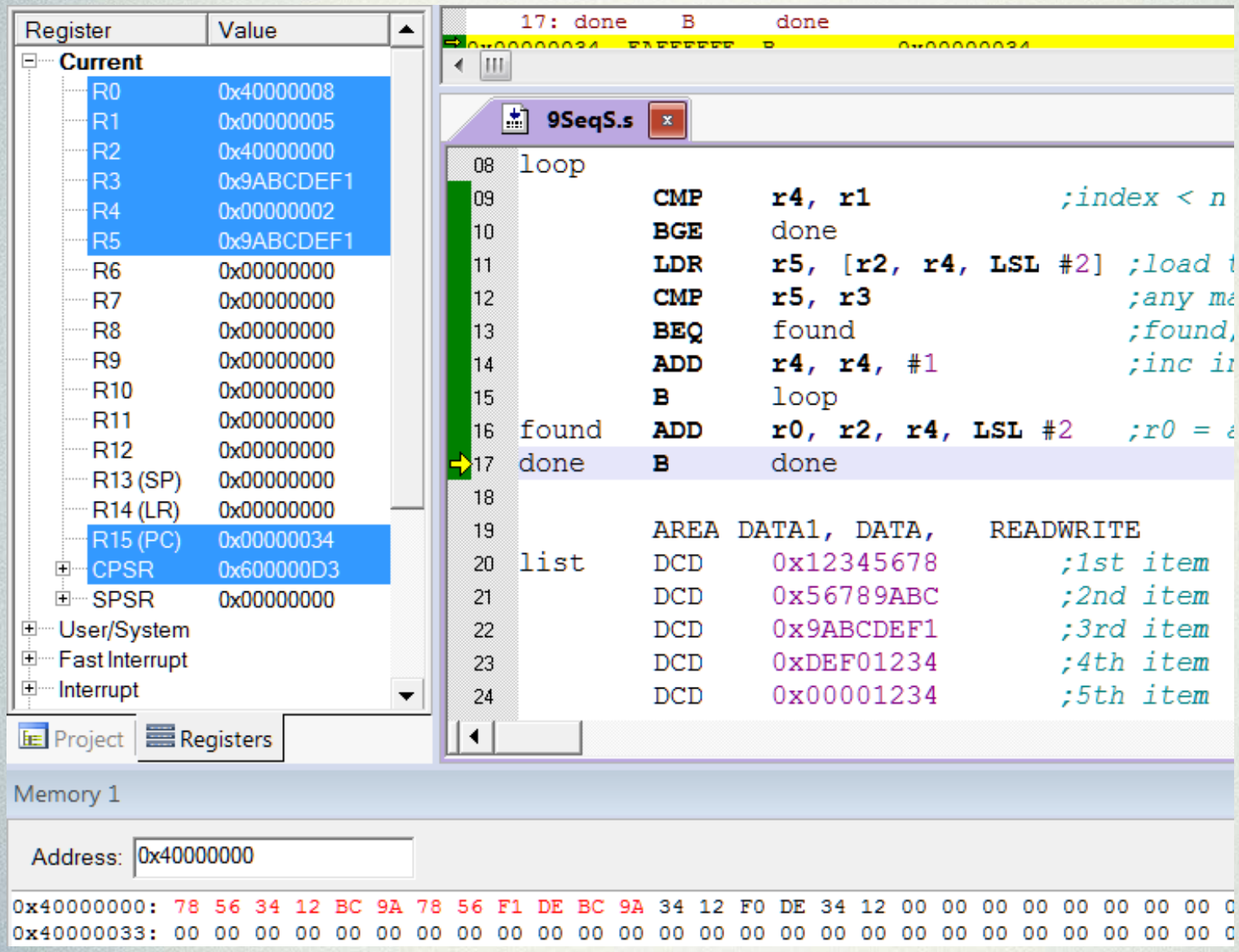
Example 9.4 Sequential Search – ARM codes 1

```
    AREA SequentialSearch, CODE, READONLY
    ENTRY
    MOV   r0, #0;found = 0 -> not found
    MOV   r1, #5;n = 5 -> number of items in list
    LDR   r2, =list      ;load starting address of list
    LDR   r3, =0x9ABCDEF1   ;load key
    MOV   r4, #0;index = 0
loop
    CMP   r4, r1  ;index < n
    BGT   done
    LDR   r5, [r2, r4, LSL #2]    ;load table item at index
    CMP   r5, r3          ;any match with the key?
    BEQ   found ;found, r0 points to matched item
    ADD   r4, r4, #1   ;inc index
    B     loop
found
    ADD   r0, r2, r4, LSL #2;r0 = address of matched item
done B    done
```

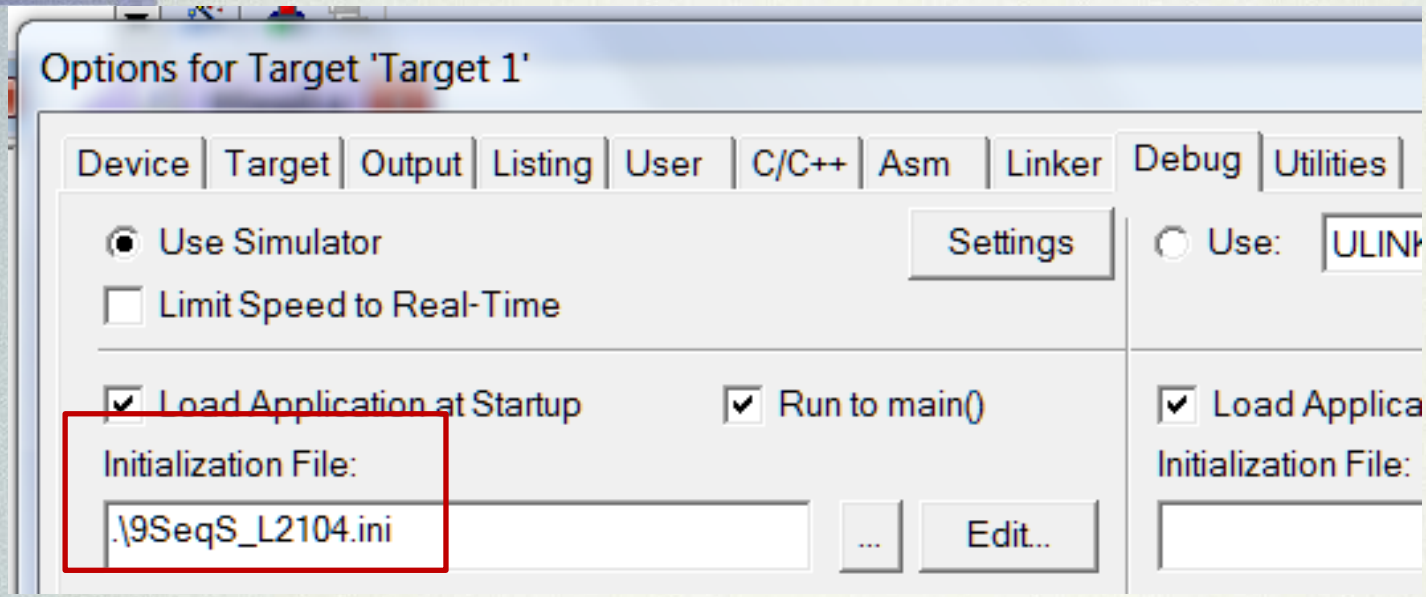
Example 9.4 Sequential Search – ARM codes 2

```
    AREA DATA1, DATA, READWRITE
list
    DCD   0x12345678 ;1st item
    DCD   0x56789ABC    ;2nd item
    DCD   0x9ABCDEF1    ;3rd item
    DCD   0xDEF01234 ;4th item
    DCD   0x00001234 ;5th item
    END
```

Example 9.4 Sequential Search – Demo



Example 9.4 Sequential Search – initial file



9SeqS.s | 9SeqS_L2104.ini

```
MAP 0x40000000, 0x40002000 EXEC READ WRITE
E LONG 0x40000000 = 0x12345678, 0x56789ABC, 0x9ABCDEF1, 0xDEF01234, 0x00001234
```

Table Summary

- Representing table/list in memory
- Accessing a table element → base + offset addressing
- Populating a table with data
  - Arithmetic progression
- Jump table → table with addresses of subroutines
- Sort and search a table with key and data
  - Bubble sort
  - Sequential search