ARM Programs

Example 1

Write program to add two nos. 32H and 40H

Store result in R7

Steps

1

2

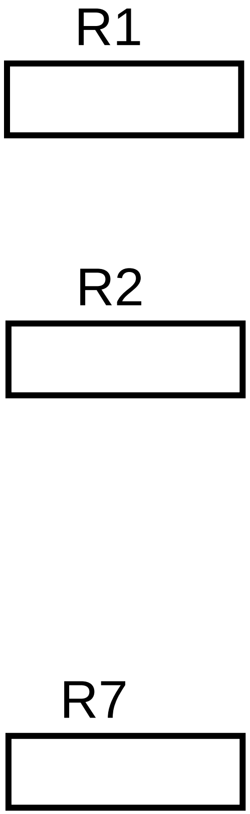
3

4

5

Example 1

Write program to add two nos. 32H and 40H, then Store result in R7



R1

R2

R7

Example 1

Write program to add two nos. 32H and 40H, then
Store result in R7

```
        AREA program, code, readonly
ENTRY
        MOV R1, #0X32
        MOV R2, #0X40
        ADD R7, R1, R2
L       B L
        END
```

Example 2

Write program to add two nos. 22H and 55H, then Store result in Memory 40000000H

Steps

1

2

3

4

5

Example 2

Write program to add two nos. 22H and 55H, then Store result in Memory 40000000H

Steps

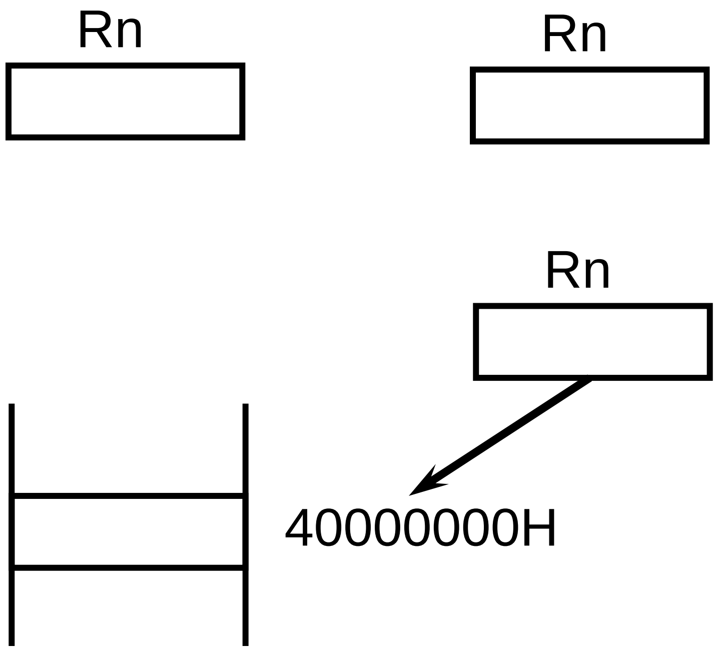
1

2

3

4

5



Example 2

Write program to add two nos. 22H and 55H, then Store result in Memory 40000000H

```
        AREA program, code, readonly
ENTRY
        MOV R1, #0X22
        MOV R2, #0X55
        MOV R3, #0X40000000
        ADD R1, R1, R2
        STR  R1, [R3]
L       B L
        END
```

Example 3

Write program to add two nos. x and y present in memory at address 4000000H and 40000004H and store in memory 40000008H

Steps

Registers required

1

2

3

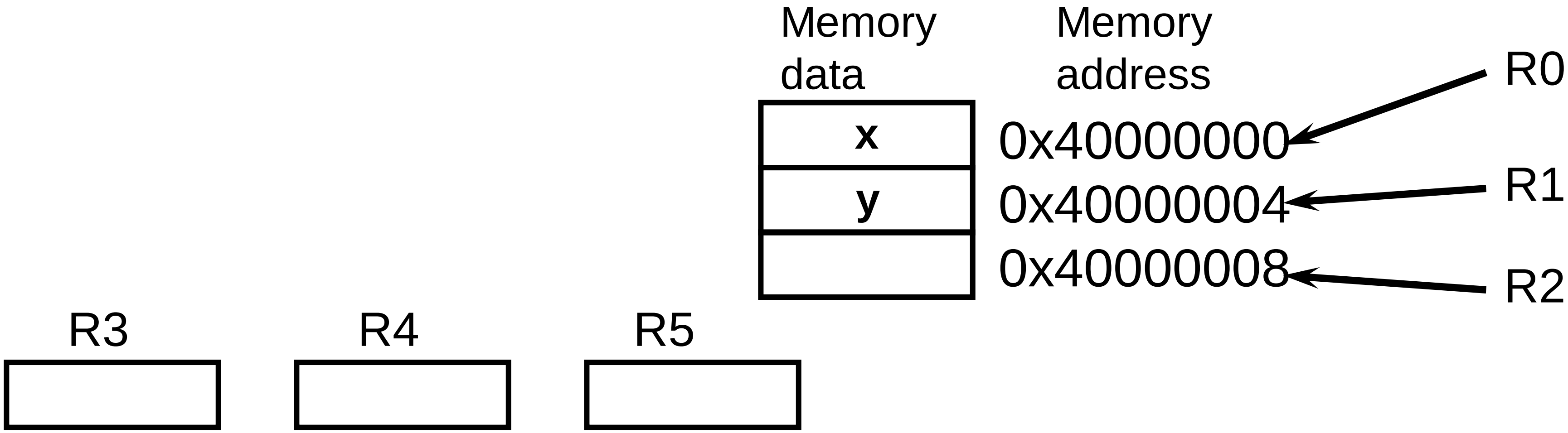
4

5

Memory data	Memory address
x	0x40000000
y	0x40000004
	0x40000008

Example 3

Write program to add two nos. x and y present in memory at address 4000000H and 40000004H and store in memory 40000008H



Example 3

Write program to add two nos. x and y present in memory at address 4000000H and 40000004H and store in memory 40000008H

- **Registers used**
- R0 = 0x40000000 ; x ptr
- R1 = 0x40000004 ; y ptr
- R2 = 0x40000008 ; dest ptr
- R3 = first operand x
- R4 = second operand y
- R5 = R3 + R4

Memory data	Memory address
x	0x40000000
y	0x40000004
	0x40000008

Example 3

- **Steps**
- R0 = 0x40000000 ; x ptr
- R1 = 0x40000004 ; y ptr
- R2 = 0x40000008 ; dest ptr
- R3 = first operand x
- R4 = second operand y
- R5 = R3 + R4
- Store result

Example 3

```
          AREA program, code, readonly
    ENTRY
                    MOV R0, #0x40000000
                    MOV R1, #0x40000004
                    ADD R2, R0, #0x08
                    LDR  R3, [R0]
                    LDR  R4, [R1]
                    ADD  R5, R3, R4
                    STR R5, [R2]
    L               B  L
                    END
```

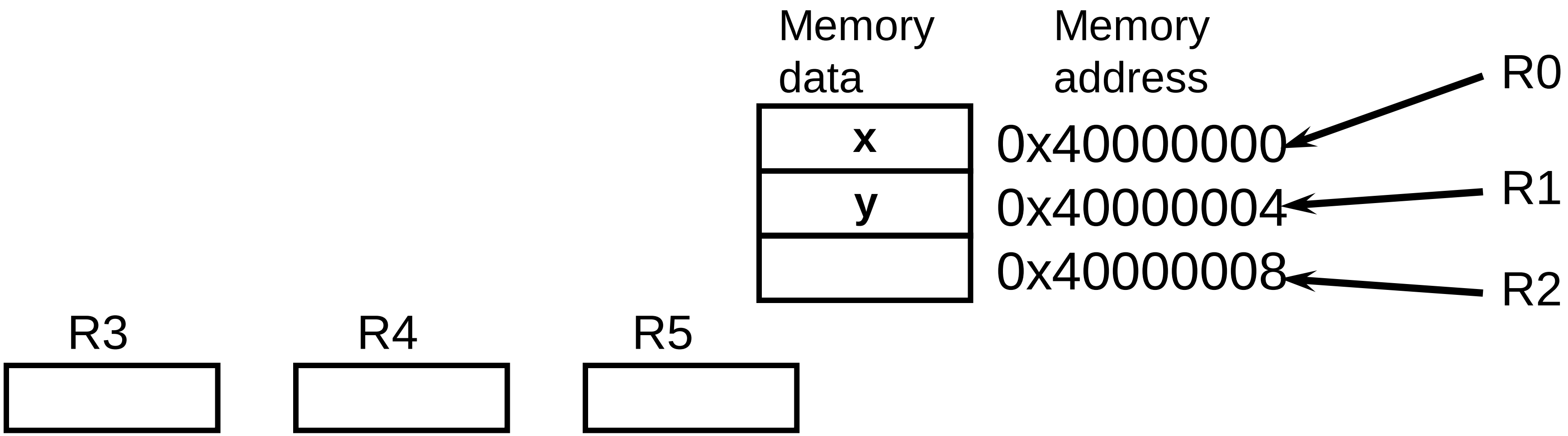
Example 1

- Write program to add two nos. x and y present in memory at address 4000000H and 40000004H and store in memory 40000008H

- **Registers used**
- R0 = 0x40000000 ; x ptr
- R1 = 0x40000004 ; y ptr
- R2 = 0x40000008 ; dest ptr
- R3 = first operand x
- R4 = second operand y
- R5 = R3 + R4

Memory data	Memory address
x	0x40000000
y	0x40000004
	0x40000008

Example 1….



Memory data

Memory address

x 0x40000000 ← R0

y 0x40000004 ← R1

0x40000008 ← R2

R3

R4

R5

```
          AREA program, code, readonly
ENTRY
          MOV R0, #0x40000000
          MOV R1, #0x40000004
          ADD R2, R0, #0x08
          LDR  R3, [R0]
          LDR  R4, [R1]
          ADD  R5, R3, R4
          STR R5, [R2]
L         B  L
          END
```

Example 2

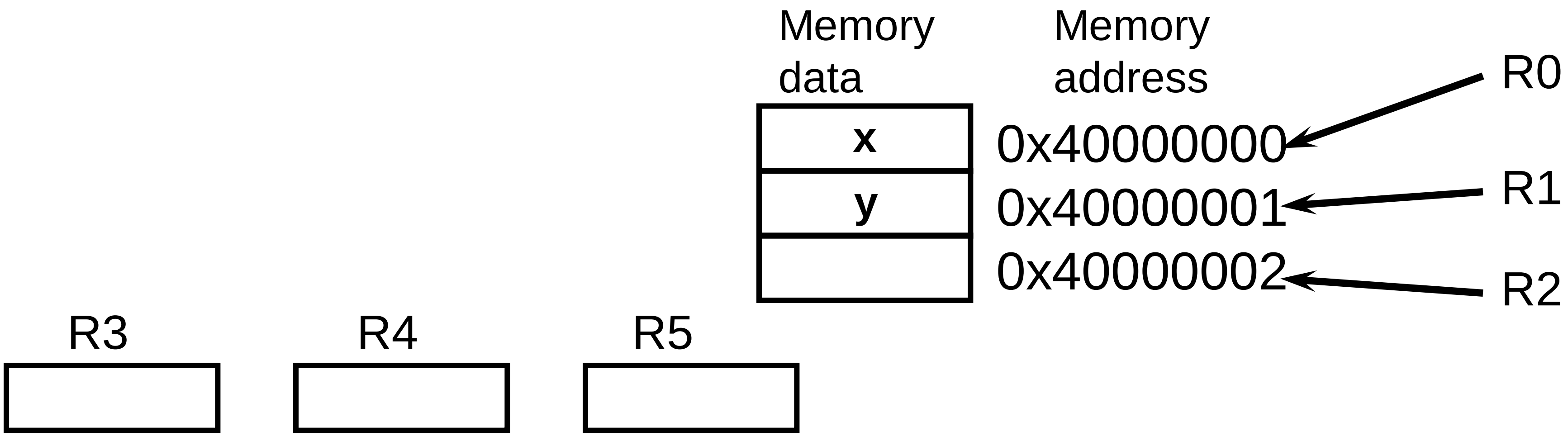
- Write program to add two 8 bit nos. x and y present in memory at address 4000000H and 40000001H and store in memory 40000002H

- **Registers used**
- R0 = 0x40000000 ; x ptr
- R1 = 0x40000001 ; y ptr
- R2 = 0x40000002 ; dest ptr
- R3 = first operand x
- R4 = second operand y
- R5 = R3 + R4

Memory data	Memory address
x	0x40000000
y	0x40000001
	0x40000002

- **For 8 bit data transfer instructions used are**
- LDRB to read data and STRB to store data

Example 2….



```
          AREA program, code, readonly
ENTRY
          MOV R0, #0x40000000
          ADD  R1, R0, #0x01
          ADD R2, R0, #0x02
          LDRB  R3, [R0]
          LDRB  R4, [R1]
          ADD  R5, R3, R4
          STRB R5, [R2]
L         B  L
          END
```

Example 3

- Write program to add 5 nos. present in memory from address 4000 0004H and store result in memory 4000 0030H

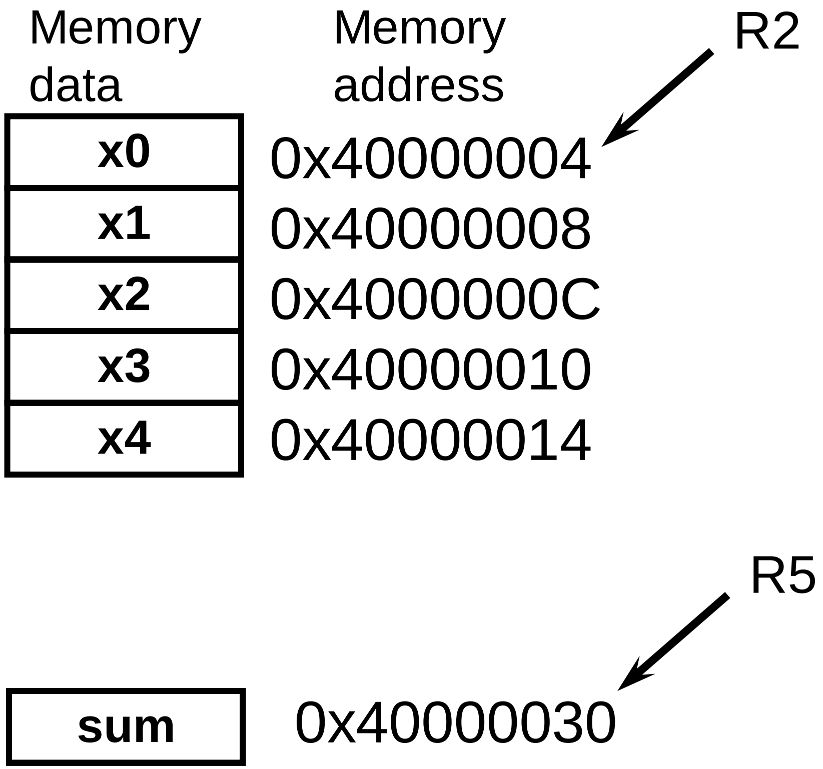
- Perform
- x0+x1+x2+x3+x4
- Store result (sum)

Memory data	Memory address
x0	0x40000004
x1	0x40000008
x2	0x4000000C
x3	0x40000010
x4	0x40000014
	0x40000030

Example 3…

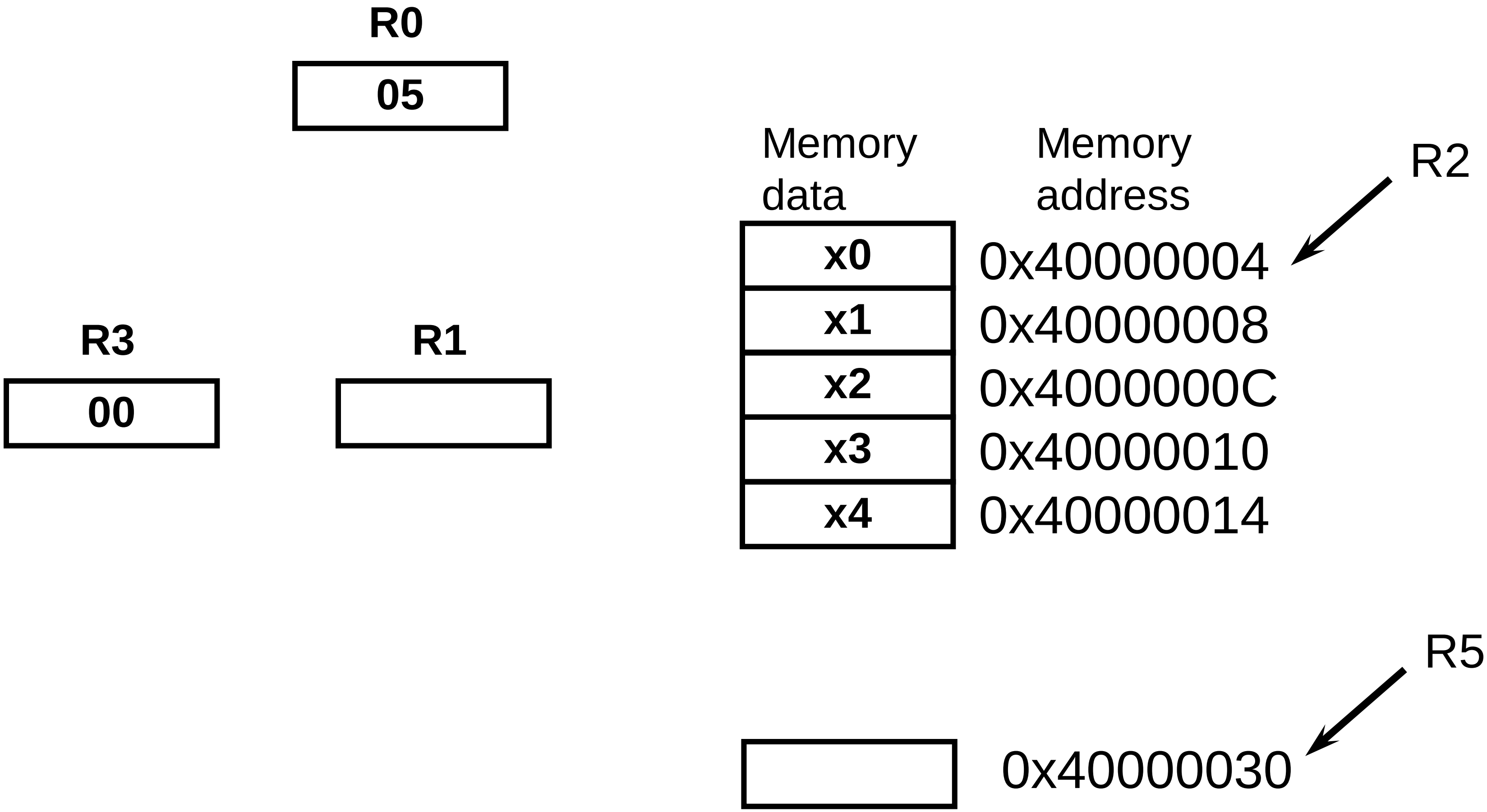
- ➢ Registers used
- ➢ R0 = 5 ; counter
- ➢ R2 = 0x40000004 ; source ptr
- ➢ R3 = 0; Sum Reg
- ➢ R1 = x

- ➢Perform
- ➢R3 = x0+x1+x2+x3+x4
- ➢R5 = 0x40000030; dest ptr
- ➢R3 = sum



Memory data	Memory address
x0	0x40000004
x1	0x40000008
x2	0x4000000C
x3	0x40000010
x4	0x40000014
sum	0x40000030

Example 3…



R0

05

R3

00

R1

Memory data	Memory address
x0	0x40000004
x1	0x40000008
x2	0x4000000C
x3	0x40000010
x4	0x40000014
	0x40000030

R2

R5

Example 3…

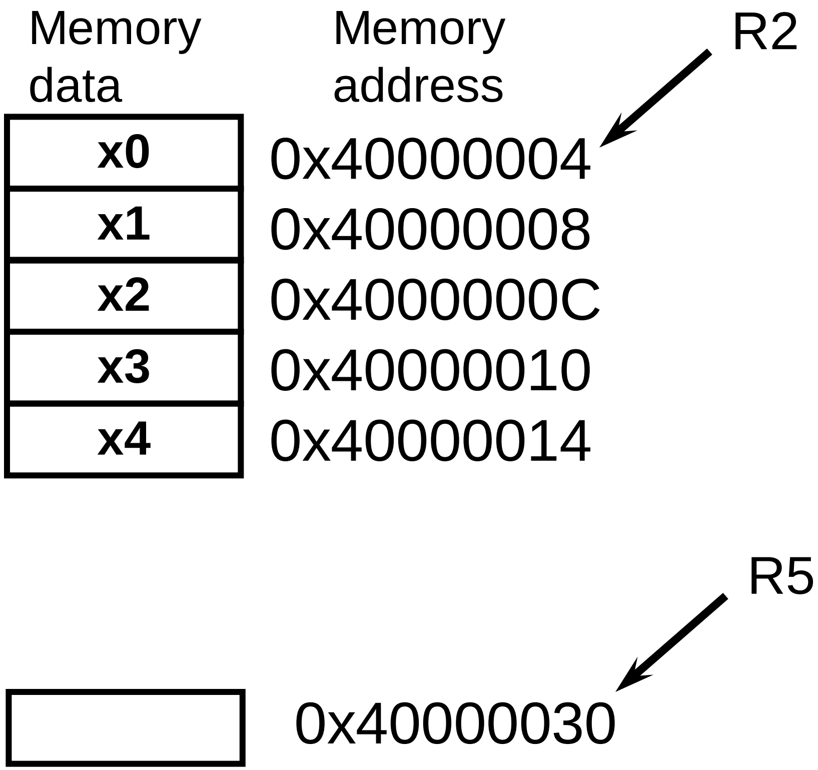
R0	R3
05	00

step 1-
R2=0X40000004, R0=5,
R3=0,
step 2-

```
LDR  R1, [R2], #4
ADD  R3, R3, R1
SUB  R0, R0, #1
```

Repeat step 2 if R0 not 00
step 3-
R5=0X40000030

```
STR R3, [R5]
```



Memory data	Memory address
x0	0x40000004
x1	0x40000008
x2	0x4000000C
x3	0x40000010
x4	0x40000014
	0x40000030

R2

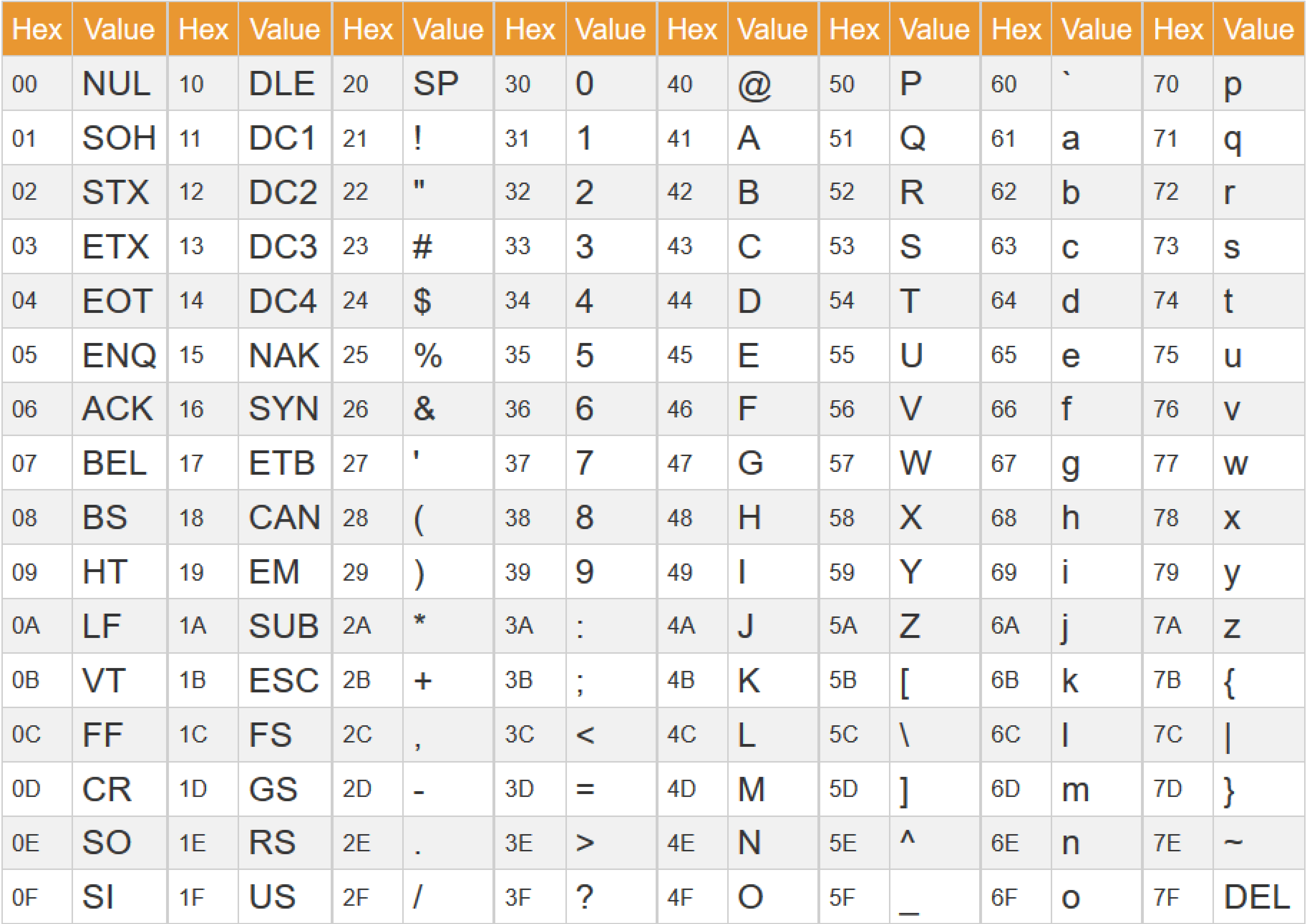
R5

Example 3….

```
          AREA program, code, readonly
ENTRY
          MOV R0, #0x0005;
          MOV R2, #0x40000004;
          MOV R3, #0x00;
NEXT      LDR R1, [R2], #4;
          ADD R3, R3, R1;
          SUB R0, R0, #01;
          CMP R0, #0x00;
          BNE NEXT
          MOV R5, #0x40000030;
          STR R3, [R5];
L         B  L
          END
```

ASCII code

- "American Standard Code for Information Interchange." (ASCII) is **a character encoding that uses numeric codes to represent characters.** These include upper and lowercase English letters, numbers, and punctuation symbols etc.



Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

ASCII code

Key	ASCII
0	30H
1	31H
2	32H
3	33H
4	34H
5	35H
6	36H
7	37H
8	38H
9	39H

Key	ASCII
A	41H
B	42H
C	43H
D	44H
E	45H
F	46H
G	47H
H	48H
Z	5AH

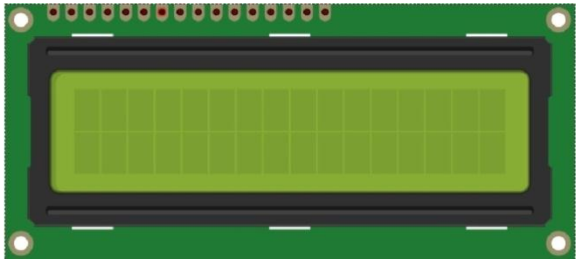
Key	ASCII
a	61H
b	62H
c	63H
d	64H
e	65H
f	66H
g	67H
h	68H
z	7AH

Microcontroller based System

Hex no.

CPU
Microcontroller

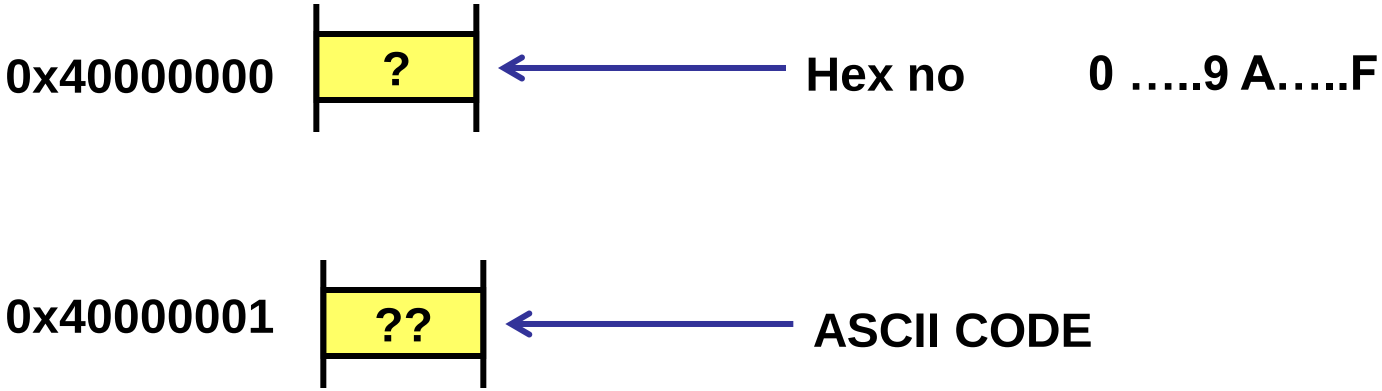
ASCII.



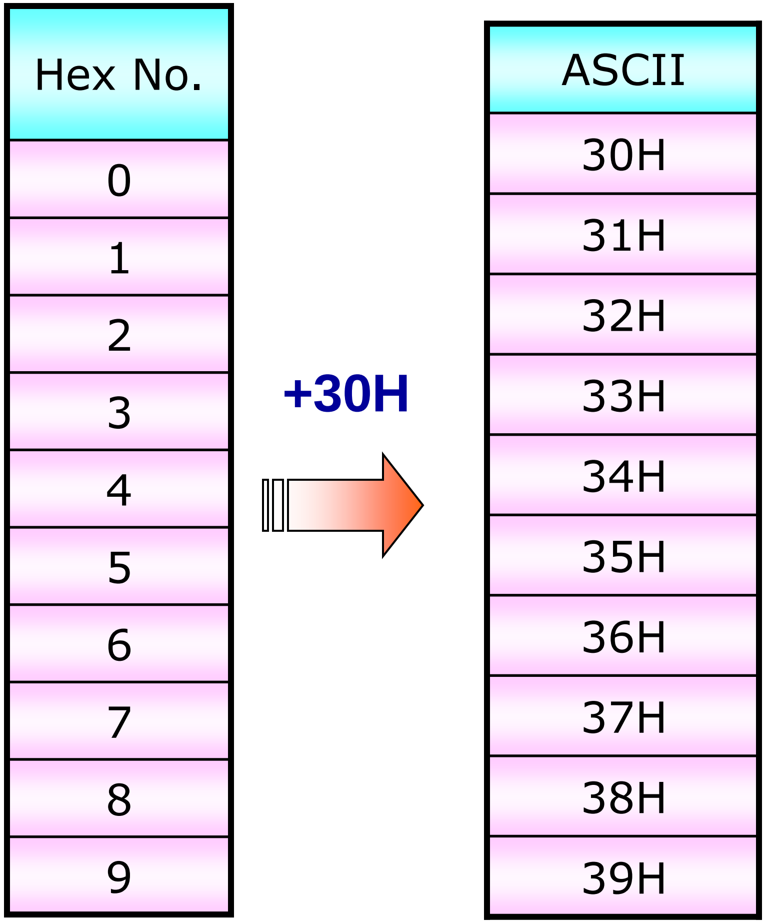
memory	Reg
A	41H

Example 4

- Write program to Convert single digit hex no into its equivalent ASCII CODE
- Assume that single digit hex no. is present in memory at address 0x40000000
- Store equivalent ASCII CODE in memory 0x40000001



Example 4..



Hex No.	ASCII
0	30H
1	31H
2	32H
3	33H
4	34H
5	35H
6	36H
7	37H
8	38H
9	39H

+30H

- ➢ **Conversion formula for 0 to 9**
- ➢ **Hex no + 30H = ASCII CODE**

Example 4…

Hex No.
A
B
C
D
E
F

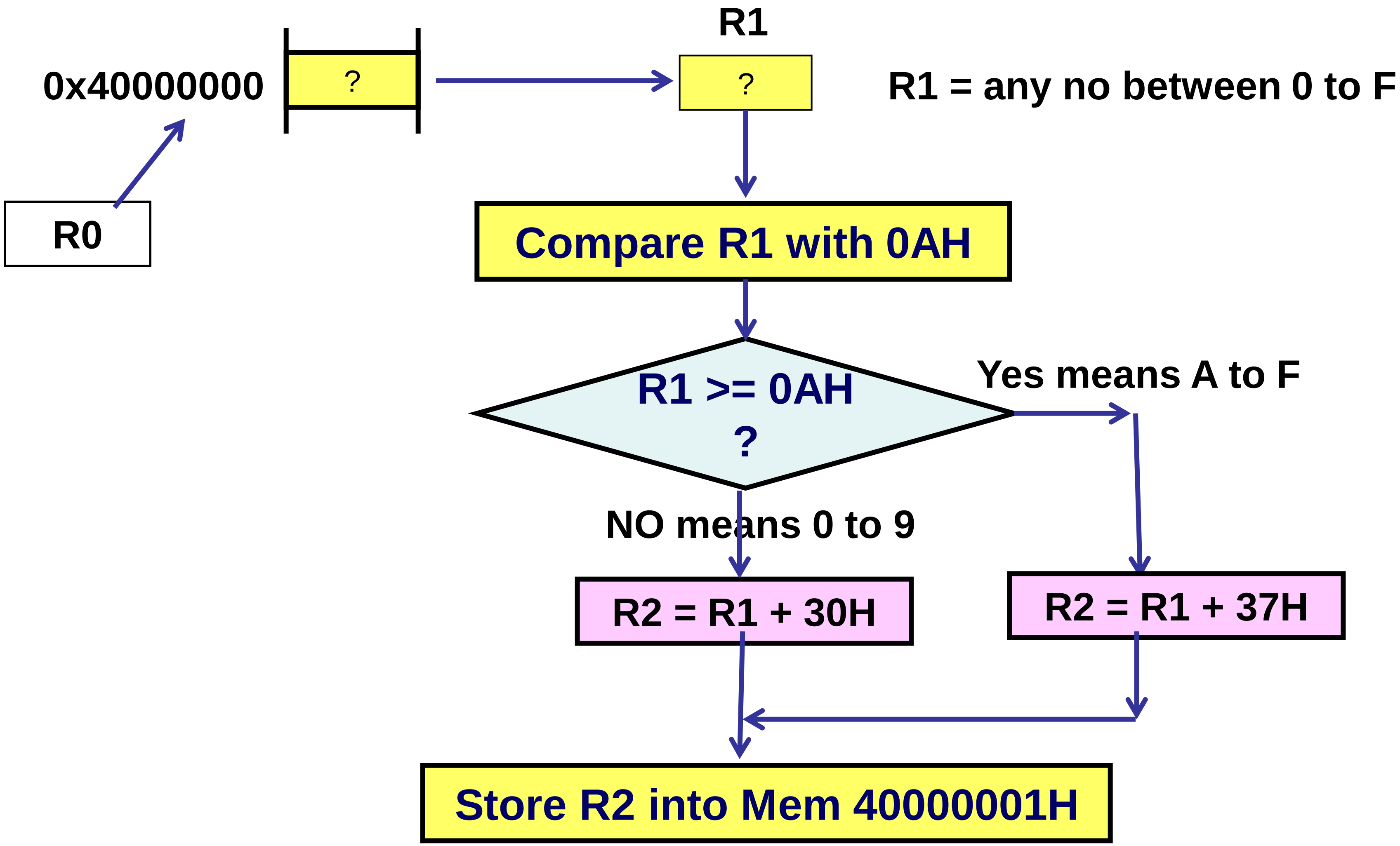
+37H

ASCII
41H
42H
43H
44H
45H
46H

- **Conversion formula for A to F**
- **Hex no + 37H = ASCII CODE**

Example 4..

- **Program Logic**



Example 4..

- **ARM Program**

```
        MOV R0,#0X40000000      ; Set Source address pointer
        ADD  R3, R0,#01         ; Set Destination address pointer
        LDRB R1,[R0]            ; Load hex no in R1
        CMP R1,#0x0A            ; Compare hex no. with 0AH
        ADDLT R2, R1,#0X30      ; If less add 30H
        ADDGE R2, R1,#0X37      ; if greater / equal add 37H
        STRB R2, [R3]           ; Store ASCII CODE in memory
L       B    L
```

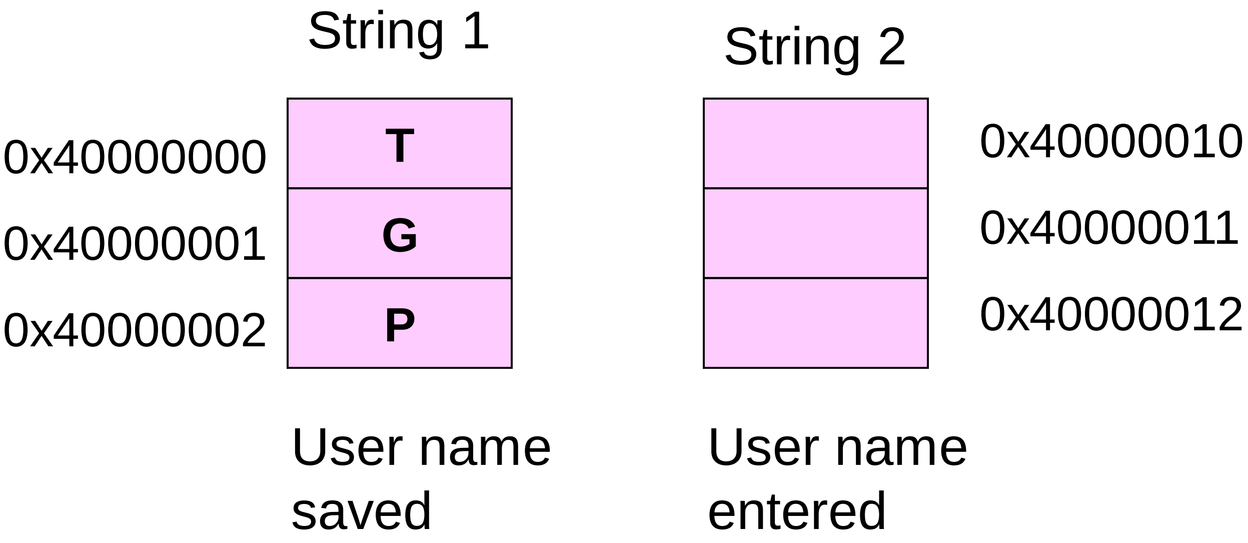
Example 4.. Run on Keil IDE

- **ARM Program**

```
          AREA program, code, readonly
ENTRY
          MOV R0,#0X40000000  ; Set Source address pointer
          ADD  R3, R0,#01     ; Set Destinationaddress pointer
          LDRB R1,[R0]        ; Load hex no in R1
          CMP R1,#0x0A        ; Compare hex no. with 0AH
          ADDLT R2, R1,#0X30  ; If less add 30H
          ADDGE R2, R1,#0X37  ; if greater / equal add 37H
          STRB R2, [R3]       ; Store ASCII CODE in memory
L   B  L                      ; stop
          END
```

Example 5

- Compare two strings of 3 ASCII character
- One string starts at 0x40000000 and other at 0x40000010.
- If both the string match store 11H in memory location 0x40000030 otherwise store FFH in memory location 0x40000030.



String 1

String 2

0x40000000 T

0x40000001 G

0x40000002 P

0x40000010

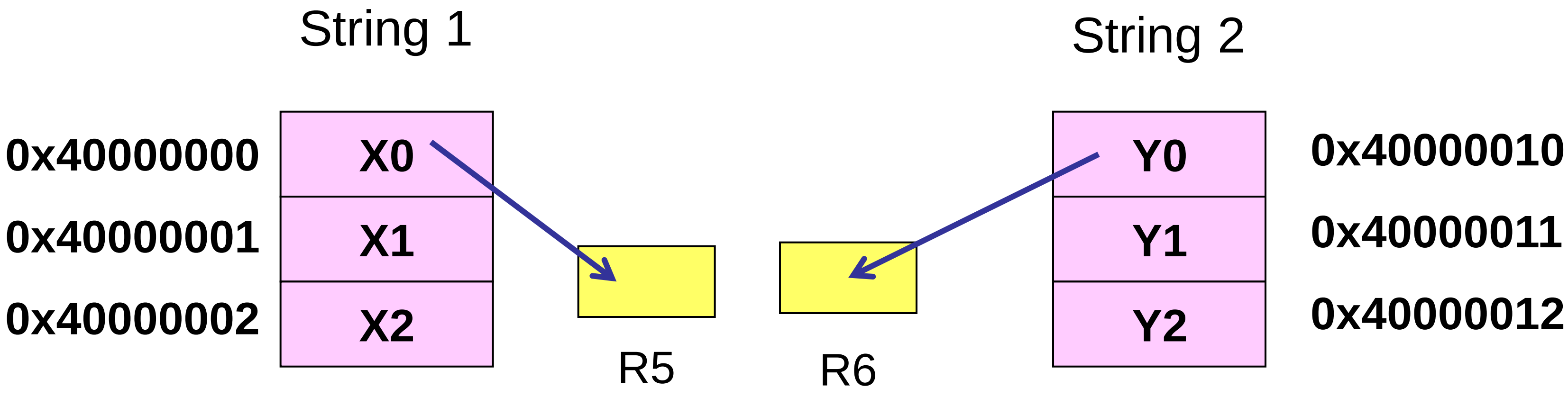
0x40000011

0x40000012

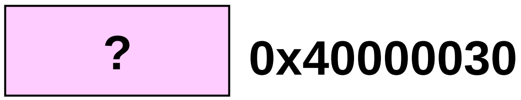
User name saved

User name entered

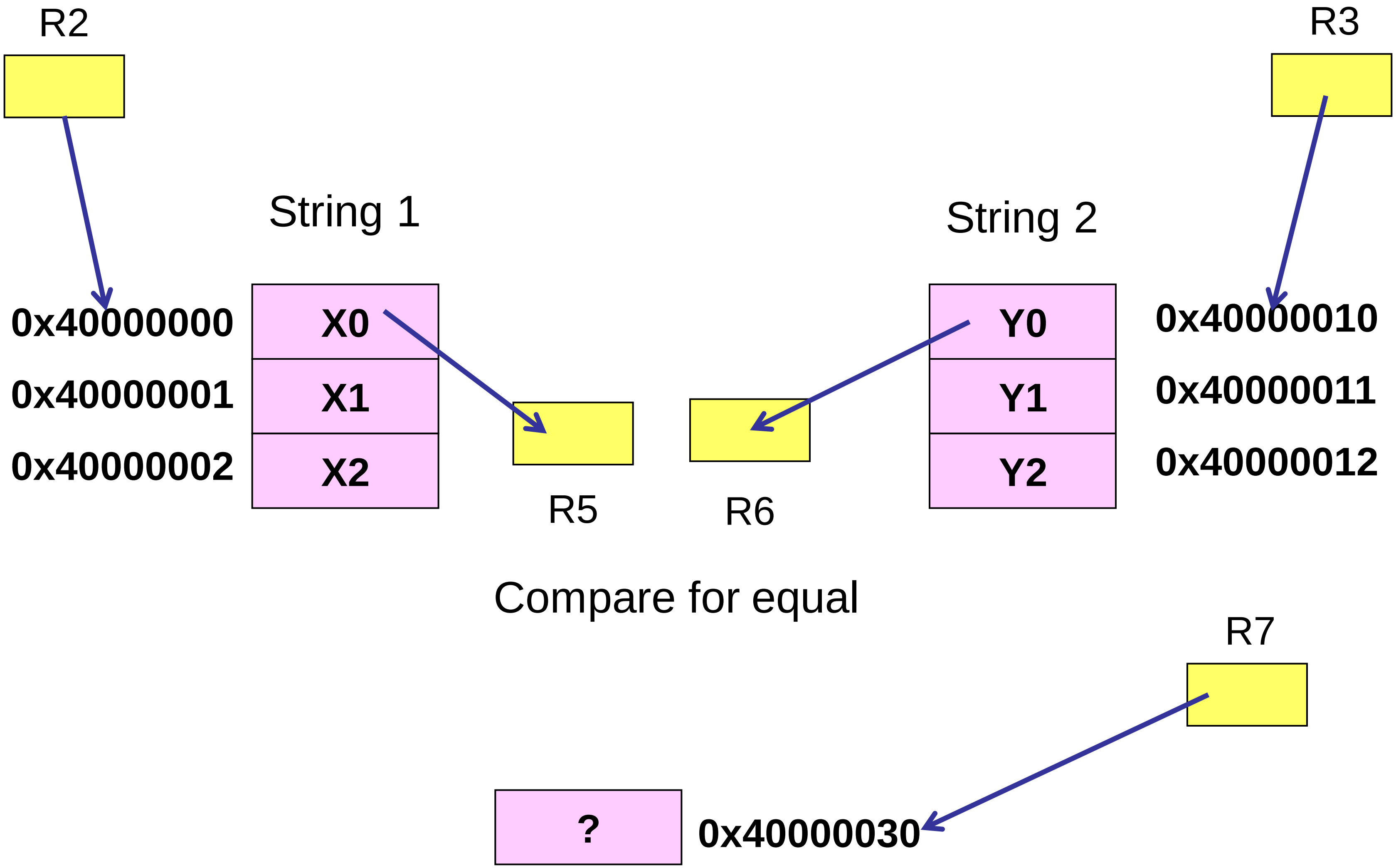
Example 5..



- Compare R5 and R6
- If R5 = R6 then Compare next character
- If all three match then store 11H to mem 0x40000030
- Else store FFH to mem 0x40000030

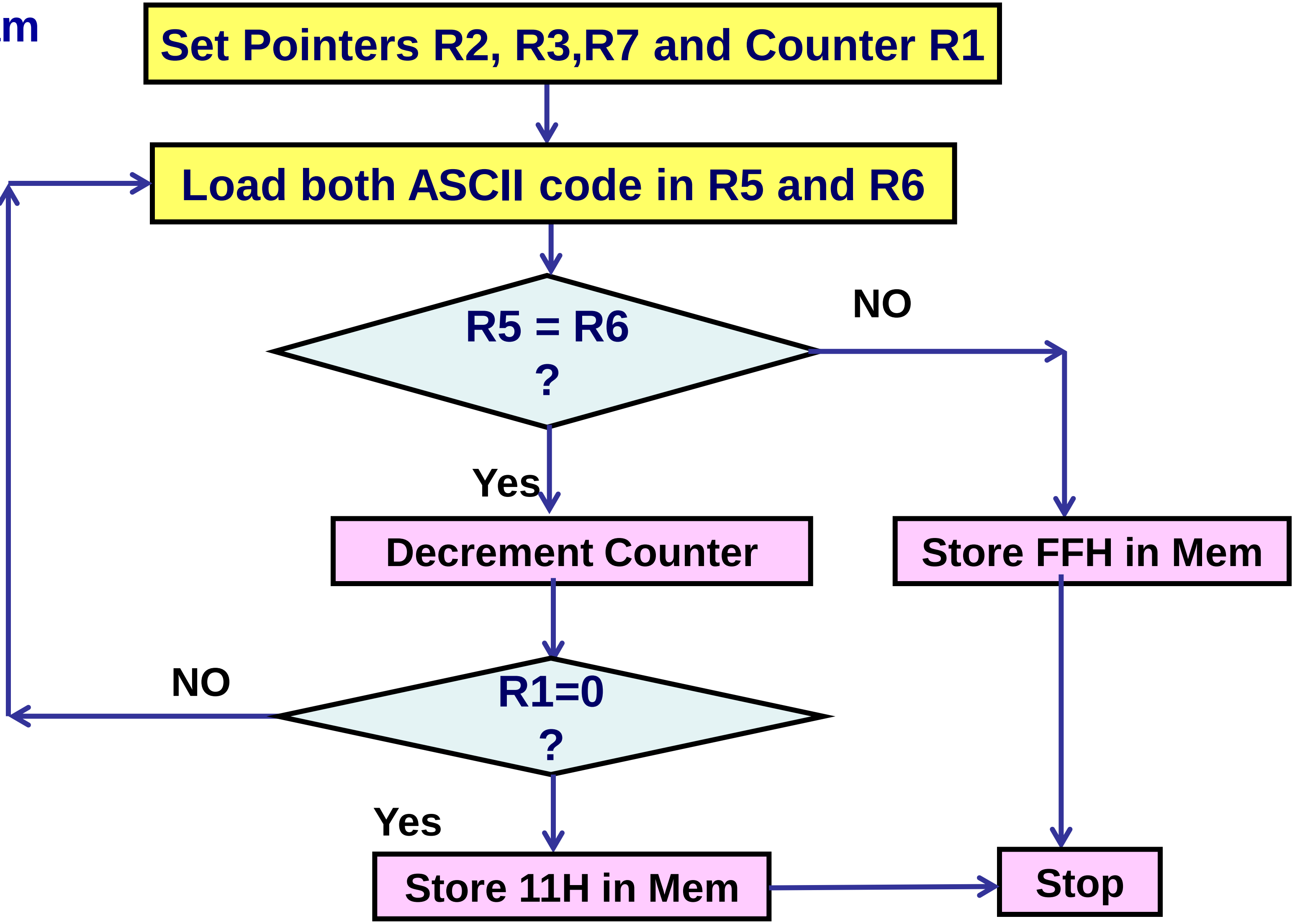


Example 5…


R2
R3
String 1
String 2
0x40000000
X0
0x40000001
X1
0x40000002
X2
Y0
0x40000010
Y1
0x40000011
Y2
0x40000012
R5
R6
Compare for equal
R7
?
0x40000030

Example 5…

- **Program Logic**



Example 5..

```
                MOV R1,#0x03            ; Counter =3
                MOV R2,#0x40000000      ; Set Pointer 1
                ADD R3, R2, #0x10       ; Set Pointer 2
                ADD R7, R2, #0x30       ; Set Pointer 3
                MOV R4,#0xFF            ; store FF in reg R4
NEXT            LDRB R5,[R2],#1         ; Load char from string 1
                LDRB R6,[R3],#1         ; load char from string 2
                CMP R5,R6               ; Compare ASCII codes
                BNE DONE                ; not equal then store FFH
                SUB R1,#01              ; Decrement counter
                CMP R1,#00              ; Check for 00
                BNE NEXT                ; Go for next char
                MOV R4,#0x11            ; R4 =11H
DONE            STRB R4,[R7]            ; Store R4 into mem
L               B    L
```

Example 5.. Run on Keil IDE

```
          AREA program, code, readonly
ENTRY
                    MOV R1,#0x03            ; Counter =3
                    MOV R2,#0x40000000      ; Set Pointer 1
                    ADD R3, R2, #0x10       ; Set Pointer 2
                    ADD R7, R2, #0x30       ; Set Pointer 3
                    MOV R4,#0xFF            ; store FF in reg R4
NEXT                LDRB R5,[R2],#1         ; Load char from string 1
                    LDRB R6,[R3],#1         ; load char from string 2
                    CMP R5,R6               ; Compare ASCII codes
                    BNE DONE                ; not equal then store FFH
                    SUB R1,#01              ; Decrement counter
                    CMP R1,#00              ; Check for 00
                    BNE NEXT                ; Go for next char
                    MOV R4,#0x11            ; R4 =11H
DONE                STRB R4,[R7]            ; Store R4 into mem
L                   B    L                  ; stop
          END
```

Example 6

- Multiply the two data word array of Ten nos. each
- First data array is stored in memory starting from 0x40000000 and other at 0x40000040.
- Multiply two array data and store result as array in memory starting from ML 0x40000080.

Example 6..

0x40000000	X0
0x40000004	X1
0x40000008	X2
0x4000000C	X3
0x40000010	X4
0x40000014	X5
0x40000018	X6
0x4000001C	X7
0x40000020	X8
0x40000024	X9

Data block 1

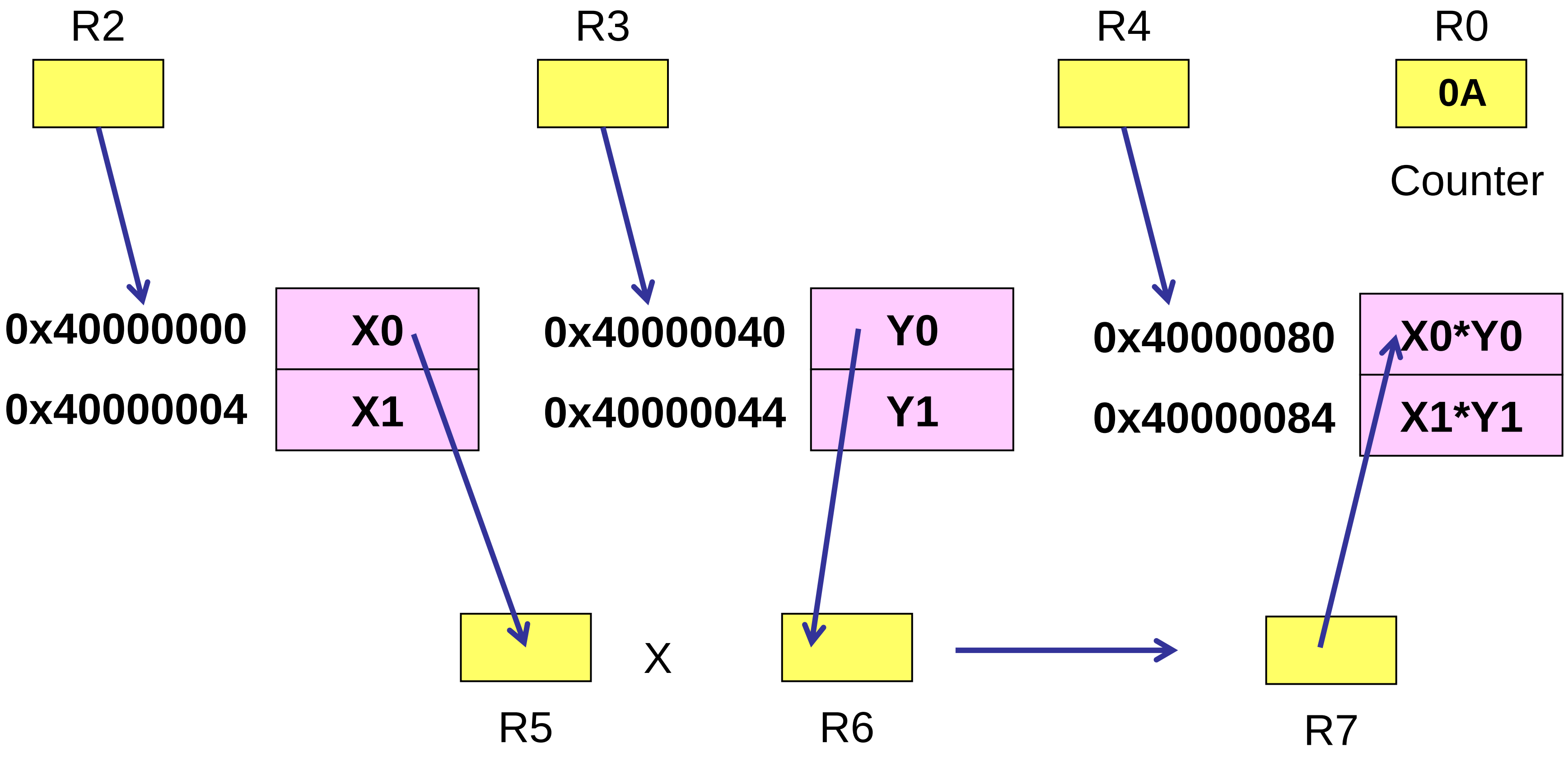
0x40000040	Y0
0x40000044	Y1
0x40000048	Y2
0x4000004C	Y3
0x40000050	Y4
0x40000054	Y5
0x40000058	Y6
0x4000005C	Y7
0x40000060	Y8
0x40000064	Y9

Data block 2

0x40000080	X0*Y0
0x40000084	X1*Y1
0x40000088	X2*Y2
0x4000008C	X3*Y3
0x40000090	X4*Y4
0x40000094	X5*Y5
0x40000098	X6*Y6
0x4000009C	X7*Y7
0x400000A0	X8*Y8
0x400000A4	X9*Y9

Destination Block

Example 6 – Program Logic



Example 6 –Program

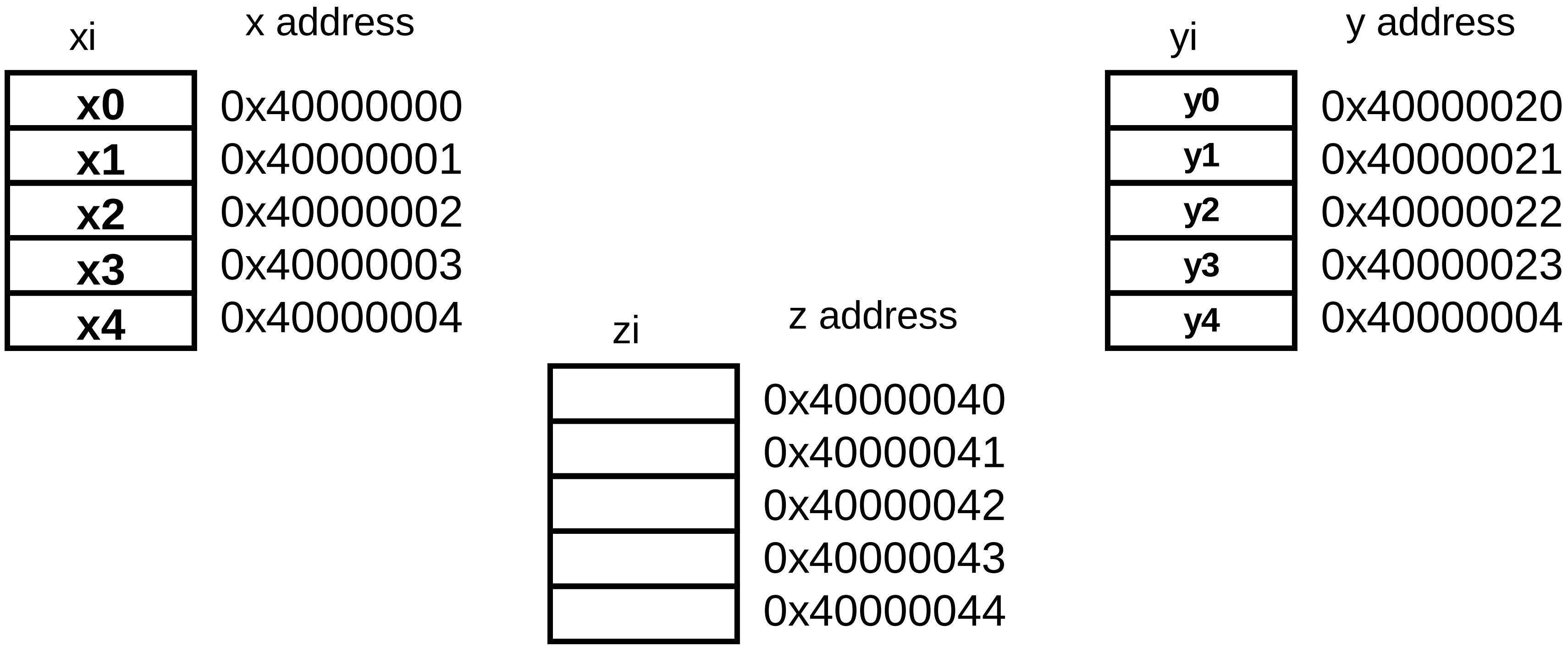
```
          MOV R0,#0X0A          ; set counter
          MOV R2,#0X40000000    ; set pointer 1
          MOV R3,#0X40000040    ; set pointer 2
          MOV R4,#0X40000080    ; set pointer 3
NEXT      LDR R5,[R2],#4        ; load data from array 1
          LDR R6,[R3],#4        ; load data from array 2
          MUL R7,R5,R6          ; multiply Xi * Yi
          STR R7,[R4],#4        ; store result in array 3
          SUB R1,#01            ; decrement counter by 1
          CMP R1,#00            ; check counter for 0
          BNE NEXT              ; Repeat till counter = 0
L         B    L
```

Example 6 –Program for Keil

```
          AREA program, code, readonly
ENTRY
          MOV  R0, #0x0A
          MOV  R2, #0x40000000
          ADD  R3, R2, #0x40
          ADD  R4, R2, #0x80
NEXT      LDR  R5, [R2], #4
          LDR  R6, [R3], #4
          MUL  R7, R5, R6
          STR R7, [R4], #4
          SUB R0, R0, #01
          CMP R0, #0x00
          BNE NEXT
L   B  L
          END
```

Example 7

- xi and yi are array of 8 bit nos
- Write program to multiply data of two array zi = xi * yi
- array1 (xi) stored from address 40000000H
- array 2 (yi) stored from address 40000020H
- no. of elements in array i = 5
- Store result array3 (zi) from address 40000040H



xi	x address
x0	0x40000000
x1	0x40000001
x2	0x40000002
x3	0x40000003
x4	0x40000004

yi	y address
y0	0x40000020
y1	0x40000021
y2	0x40000022
y3	0x40000023
y4	0x40000004

zi	z address
	0x40000040
	0x40000041
	0x40000042
	0x40000043
	0x40000044

Example 7…

Xi	x address
x0	0x40000000
x1	0x40000001
x2	0x40000002
x3	0x40000003
x4	0x40000004

Yi	y address
y0	0x40000020
y1	0x40000021
y2	0x40000022
y3	0x40000023
y4	0x40000024

zi	z address
	0x40000040
	0x40000041
	0x40000042
	0x40000043
	0x40000044

- Registers used
- R0 = 0x05 ; counter
- R1 = 0x40000000 ; x ptr
- R2 = 0x40000020 ; y ptr
- R3 = 0x40000040 ; z ptr
- R4 =0 ; for result

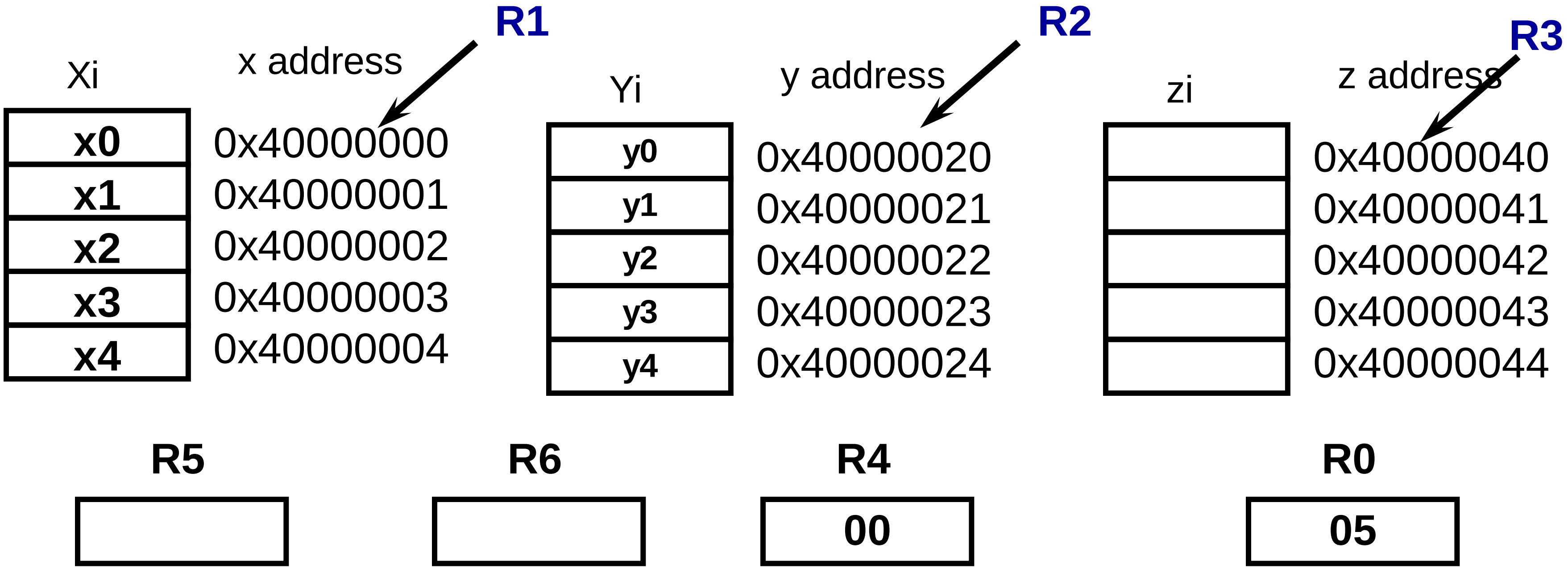
R5 = xi ; load xi

R6 = yi ; load yi

Program---

1. R4 = R5 * R6 = xi *yi = zi
2. Store R4 to memory [R3]
3. Repeat step 1, 5 times

Example 7…



R1

Xi	x address
x0	0x40000000
x1	0x40000001
x2	0x40000002
x3	0x40000003
x4	0x40000004

R2

Yi	y address
y0	0x40000020
y1	0x40000021
y2	0x40000022
y3	0x40000023
y4	0x40000024

R3

zi	z address
	0x40000040
	0x40000041
	0x40000042
	0x40000043
	0x40000044

R5	R6	R4	R0
		00	05

Example 7…

- Write program to multiply data of two array zi = xi * yi

```
         AREA program, code, readonly
ENTRY
         MOV  R0, #0x05;
         MOV  R1, #0x40000000
         ADD  R2, R1,  #0x20
         ADD  R3, R1,  #0x40
NEXT     LDRB  R5, [R1], #1
         LDRB  R6, [R2], #1
         MUL  R4, R5, R6
         STRB  R4, [R3], #1
         SUB R0, R0, #01
         CMP R0, #0x00
         BNE NEXT
L        B  L
         END
```

Q. Write program to STMFD

```
                    AREA program, code,
readonly
ENTRY
          MOV R13, #0X40000000
          ADD R13, R13, #0X30
          MOV R0, #0X10
          MOV R1, #0X20
          MOV R2, #0X30
          MOV R3, #0X40
          MOV R4, #0X50
          STMFD  SP!, {R0- R4}
L         B  L
          END
```

Q. Write program to push and pop data of reg R0-R4 in Full Descending Stack

```
AREA program, code, readonly
ENTRY
            MOV R13, #0X40000000
            ADD R13, R13, #0X30
            MOV R0, #0X10
            MOV R1, #0X20
            MOV R2, #0X30
            MOV R3, #0X40
            MOV R4, #0X50
            STMFD  SP!, {R0- R4}
            MOV R0, #0X00
            MOV R1, #0X00
            MOV R2, #0X00
            MOV R3, #0X00
            MOV R4, #0X00
            LDMFD  SP!, {R0- R4}
L           B  L
            END
```

Q. Write program to push and pop data of reg R0-R4 in Full Ascending Stack

```
        AREA program, code, readonly
ENTRY
        MOV R13, #0X40000000
        ADD R13, R13, #0X30
        MOV R0, #0X10
        MOV R1, #0X20
        MOV R2, #0X30
        MOV R3, #0X40
        MOV R4, #0X50
        STMFA  SP!, {R0- R4}
        EOR  R0, R0
        EOR  R1, R1
        EOR  R2, R2
        EOR  R3, R3
        EOR  R4, R4
        LDMFA  SP!, {R0- R4}
L       B  L
        END
```

Q. Write program to push and pop data of reg R0-R4 in Full Descending Stack

```
        AREA program, code, readonly
ENTRY
        MOV R13, #0X40000000
        ADD R13, R13, #0X30
        MOV R0, #0X10
        MOV R1, #0X20
        MOV R2, #0X30
        MOV R3, #0X40
        MOV R4, #0X50
        STMFA  SP!, {R0- R4}
        EOR  R0, R0
        EOR  R1, R1
        EOR  R2, R2
        EOR  R3, R3
        EOR  R4, R4
        LDMFA  SP!, {R0- R4}
L       B  L
        END
```

Q. Write program to push and pop data of reg R0-R4 in Full Ascending Stack

```
        AREA program, code, readonly
ENTRY
        MOV R13, #0X40000000
        ADD R13, R13, #0X30
        MOV R0, #0X10
        MOV R1, #0X20
        MOV R2, #0X30
        MOV R3, #0X40
        MOV R4, #0X50
        STMFD  SP!, {R0- R4}
        EOR  R0, R0
        EOR  R1, R1
        EOR  R2, R2
        EOR  R3, R3
        EOR  R4, R4
        LDMFD  SP!, {R0- R4}
L       B  L
        END
```

Q. Write program and also subroutine to explain push and pop data of reg R0-R4

```
              AREA program, code, readonly
ENTRY
              MOV R13, #0X40000000
              ADD R13, R13, #0X30
              MOV R0, #0X10
              MOV R1, #0X20
              MOV R2, #0X30
              MOV R3, #0X40
              MOV R4, #0X50
              BL  sub_addr
              MOV R6, R4
L             B  L
sub_addr  STMFD  SP!, {R0- R4}
              MOV R0, #0X60
              MOV R1, #0X70
              MOV R2, #0X80
              MOV R3, #0X88
              MOV R4, #0X90
              LDMFD  SP!, {R0- R4}
              MOV PC, LR
              END
```

Q. Write program to Convert single digit hex no into its equivalent ASCII CODE. Assume that single digit hex no. is present in memory at address 0x40000000. Store equivalent ASCII CODE in memory 0x40000001

```
AREA program, code, readonly
ENTRY
            MOV R0,#0X40000000      ; Set
Source address pointer
            ADD  R3, R0,#01 ; Set
Destination address pointer
            LDRB R1,[R0]                ; Load
hex no in R1
            CMP R1,#0x0A                ;
Compare hex no. with 0AH
            ADDLT R2, R1,#0X30      ; If less
add 30H
            ADDGE R2, R1,#0X37      ; if
greater / equal add 37H
            STRB R2, [R3]               ; Store
ASCII CODE in memory
L           B  L                        ; stop
            END
```