Microprocessors

Tuba Ayhan

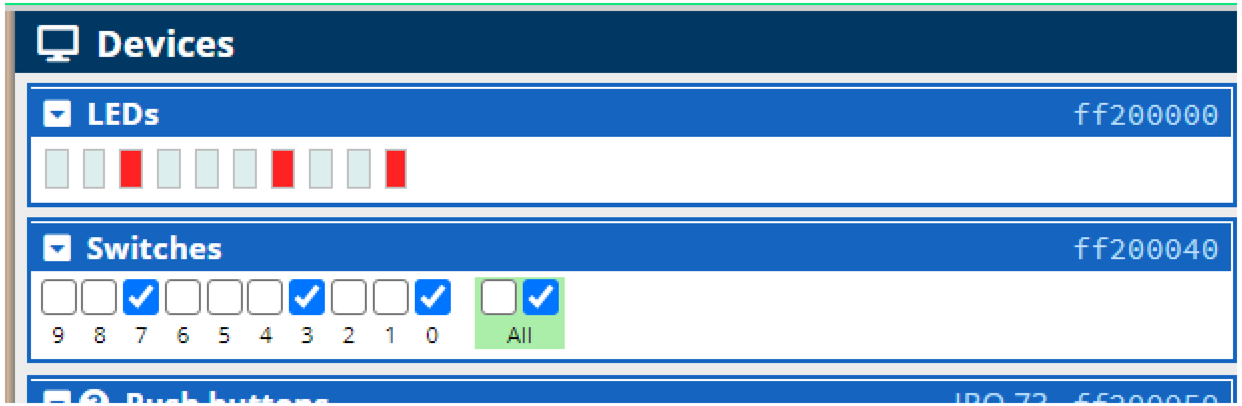
MEF University

Examples

DE1-SoC Computer System with ARM Cortex-A9

Simple I/O

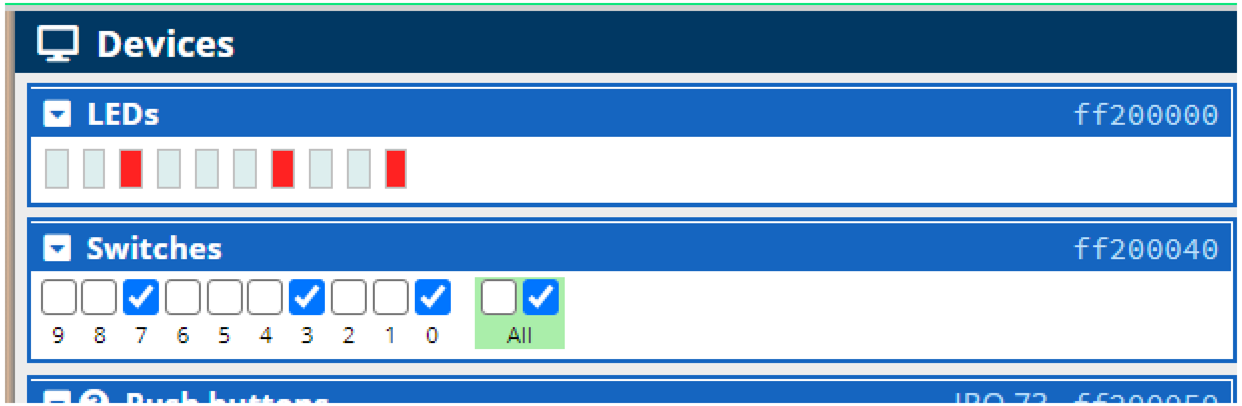
- Echo the SWs on LEDs.



Simple I/O

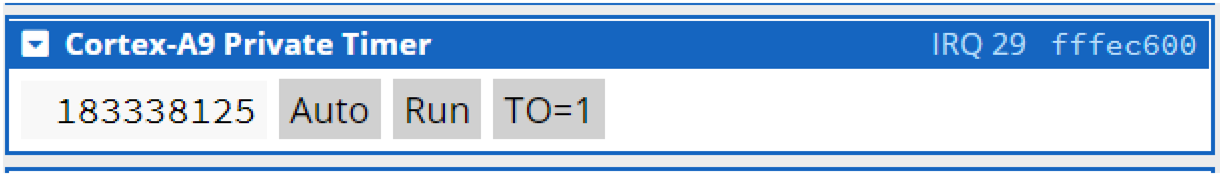
- **Echo the SWs on LEDs.**

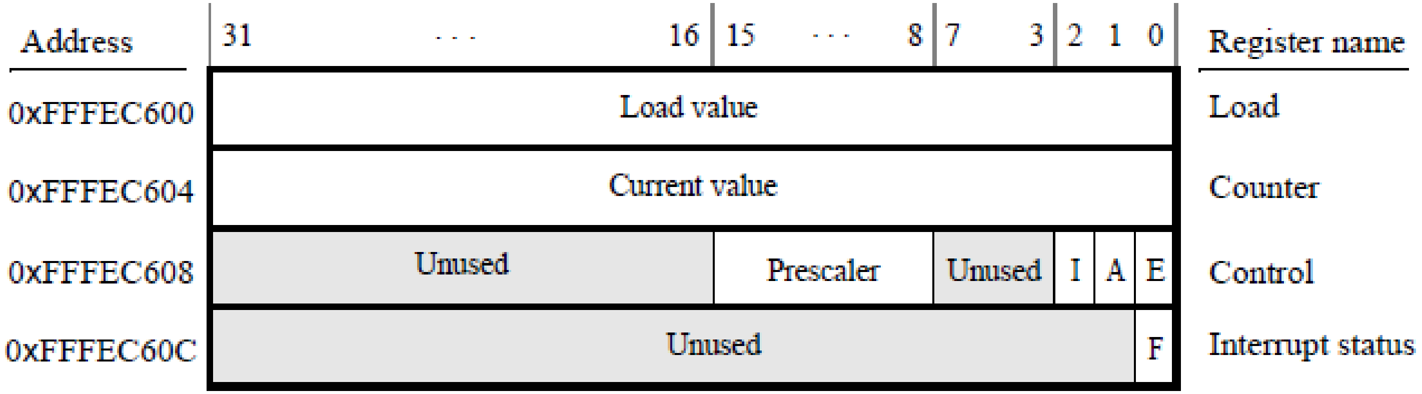
```
.global _start
_start:

        LDR R0,=0xFF200000  // R0 points to LEDs
        LDR R1,=0xFF200040  // R1 points to SWs

loop:
        LDR R3,[R1]
        STR R3,[R0]
        B loop

        .end
```



Set the private timer

- Set the timer to count for 1 second and start over when it reaches 0.
- Clk frequency = 200 MHz.

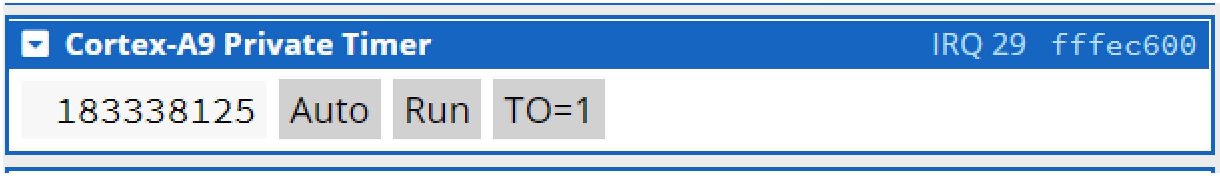




Set the private timer

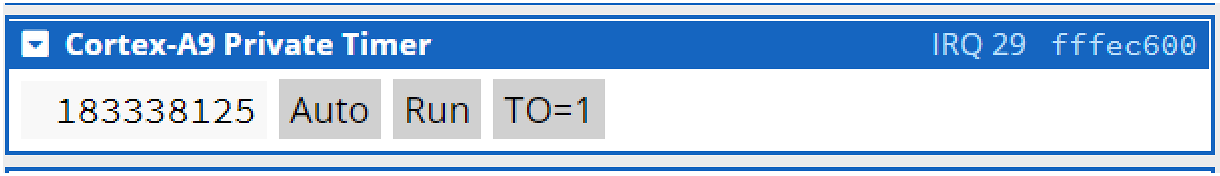
- Set the timer to count for 1 second and start over when it reaches 0.
- Clk frequency = 200 MHz.

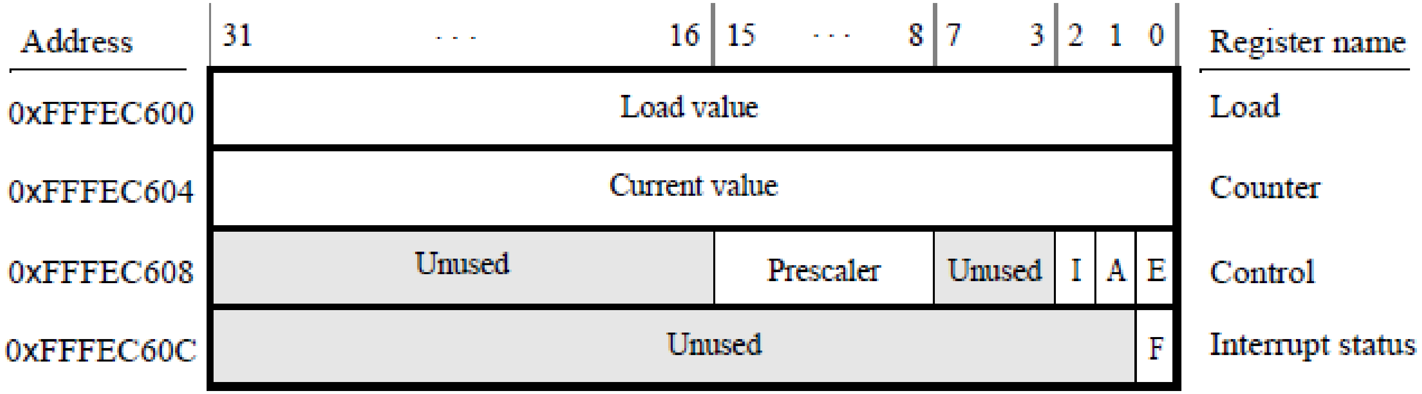
```
LDR R2,=0xFFFEC600
// R2 points to private
//timer base

        LDR R3,=200000000
        STR R3, [R2]
        MOV R3,#0b011
        STR R3, [R2,#8]
```



Address	31 ... 16	15 ... 8	7 3	2	1	0	Register name
0xFFFEC600	Load value						Load
0xFFFEC604	Current value						Counter
0xFFFEC608	Unused	Prescaler	Unused	I	A	E	Control
0xFFFEC60C	Unused					F	Interrupt status

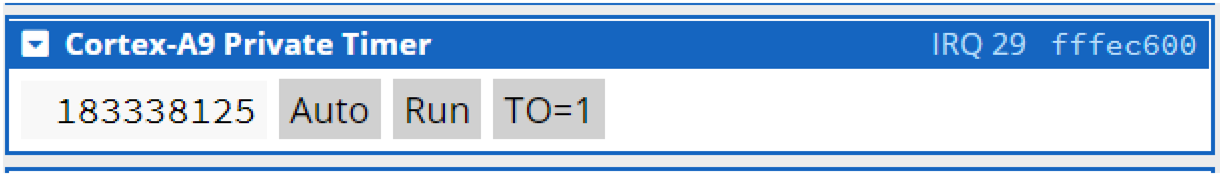
Display the private timer current value

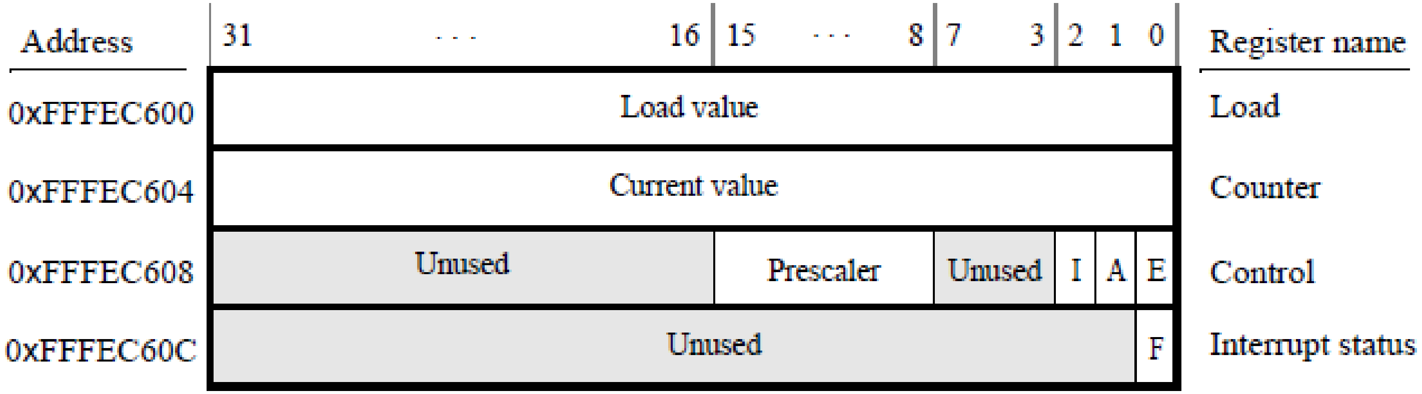




Display the private timer current value

```
loop:
    LDR R3,[R2,#4]
    STR R3,[R0]
    B loop
```





Blink the LEDs

Address	31 . . . 16	15 . . . 8	7 3	2	1	0	Register name
0xFFFEC600	Load value						Load
0xFFFEC604	Current value						Counter
0xFFFEC608	Unused	Prescaler	Unused	I	A	E	Control
0xFFFEC60C	Unused					F	Interrupt status

Blink the LEDs

```
LDR R0,=0xFF200000  // R0 points to LEDs
LDR R1,=0xFF200040  // R1 points to SWs
LDR R2,=0xFFFEC600      // R2 points to private timer base
MOV R4,#3
LDR R3,=200000000
STR R3, [R2]
MOV R3,#0b011
STR R3, [R2,#8]
loop:  ..... →
```

Address	31 ... 16	15 ... 8	7 ... 3	2	1	0	Register name
0xFFFEC600	Load value						Load
0xFFFEC604	Current value						Counter
0xFFFEC608	Unused	Prescaler	Unused	I	A	E	Control
0xFFFEC60C	Unused					F	Interrupt status

```
loop:
        STR R4,[R0]

wait1sec:
        LDR R3,[R2,#12]
        CMP R3,#1
        BNE wait1sec

        STR R3, [R2,#12]
        EOR R4, R4, #3
        B loop
```

64-bit number comparison

- Write a compare routine to compare two unsigned 64-bit values.

a) The numbers are already in general purpose registers, use minimum number of instructions. (Hint: use conditional execution.)

Num1	R3	R1
Num2	R4	R2
	High order	Low order

b) The numbers are in memory, avoid memory access if not required.

Num1:	High order
	Low order
Num2:	High order
	Low order

64-bit number comparison

a) The numbers are already in general purpose registers, use minimum number of instructions. (Hint: use conditional execution.)

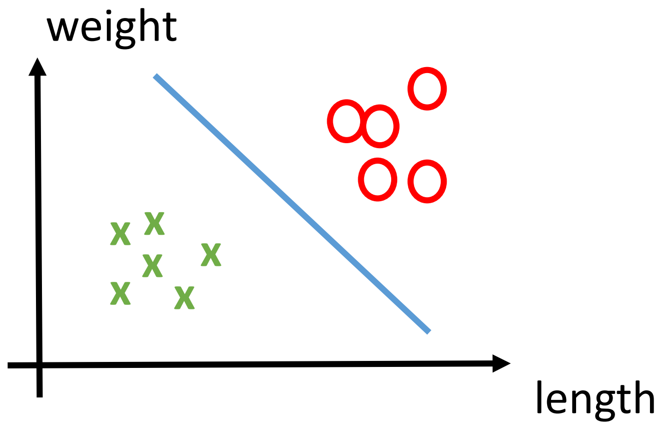
Num1	R3	R1
Num2	R4	R2
	High order	Low order

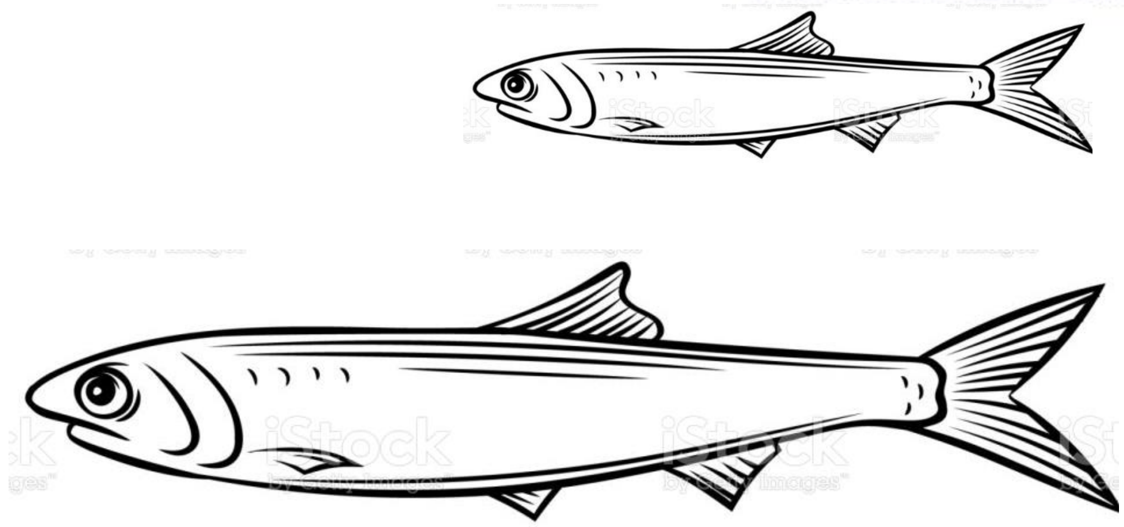
Solution

```
ldr r1, =num1
  ldr r2, =num2

  ldr r3, [r1]
  ldr r4, [r2]

  ldr r1, [r1,#4]
  ldr r2, [r2,#4]

  cmp    r3, r4
  cmpeq r1,r2
  movlt  r0,  #2
  movge  r0,  #1

  end: b end


  num1: .word 0x1fffff0f, 0x1ffffff0
  num2: .word 0x1fffff0f, 0x1ffffff8
```

```
ldr r1, =num1
   ldr r2, =num2
   ldr r3, [r1]
   ldr r4, [r2]
   cmp    r3, r4
   ldreq r1, [r1,#4]
   ldreq r2, [r2,#4]
   cmpeq  r1,r2
   movlt  r0,  #2
   movge  r0,  #1
```

Introduction to machine learning



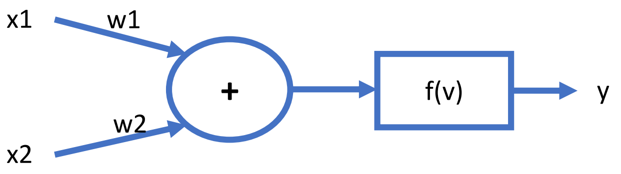


Artificial neuron

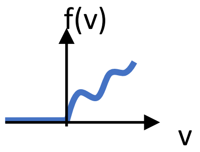
- Write a program for **an artificial neuron: y = f(v)**.

v = x1.w1 + x2.w2 , w1 = 192, w2 = -2.

Calculate “v”, but do not use multiplication instructions.

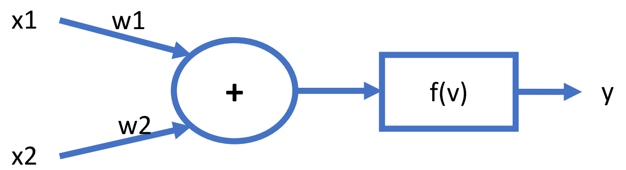


f(v) is a nonlinear function, as in the figure.
f(v) is 0, if v is negative.
Else, it is a trigonometric function.
Use a “look-up table” to calculate f(v)



x1 and x2 are memory-mapped input, located at Input1 and Input2. Read the input data, when it is ready. Calculate “y” and display the result, if the display device is not busy.

Artificial neuron

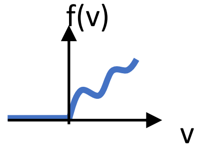


- Write a program for **an artificial neuron: y = f(v)**.

v = x1.w1 + x2.w2 , w1 = 192, w2 = -2.

Calculate "v", but do not use multiplication instructions.

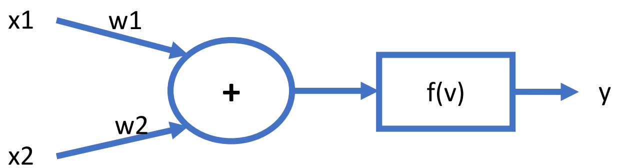
f(v) is a nonlinear function, as in the figure.
f(v) is 0, if v is negative.
Else, it is a trigonometric function.
Use a "look-up table" to calculate f(v)



x1 and x2 are memory-mapped input, located at Input1 and Input2. Read the input data, when it is ready. Calculate "y" and display the result, if the display device is not busy.

Input1:		*data*	I/O device
		status	
		control	
Input2:		*data*	I/O device
		status	
		control	
Display:		*data*	I/O device
		status	
		control	
f:	f(0)		Look-up table
	f(1)		
	f(2)		
	f(3)		
	…		

Artificial neuron 1



- Write a program for **an artificial neuron: y = f(v)**.

v = x1.w1 + x2.w2 , w1 = 192, w2 = -2.

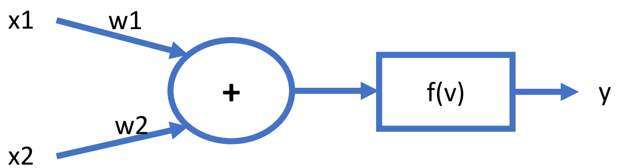
Calculate “v”, but do not use multiplication instructions.

// x1 is in R1 and x2 is in R2

// write the result v in R0

Calcv:

Artificial neuron 1



- Write a program for **an artificial neuron: y = f(v)**.

v = x1.w1 + x2.w2 , w1 = 192, w2 = -2.

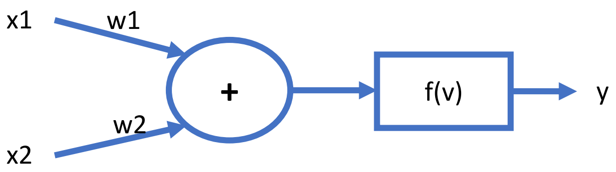
Calculate “v”, but do not use multiplication instructions.

// x1 is in R1 and x2 is in R2

// write the result v in R0

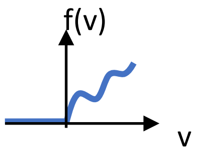
```
Calcv:  MOV R1, R1, LSL #6      // 64.x1
        ADD R1, R1, LSL #1      // 64.x1 +  64.2.x1
        MVN R2, R2, LSL #1      // -2.x2-1
        ADD R2,R2,#1            // -2.x2
        ADD R0,R1,R2            // v
```

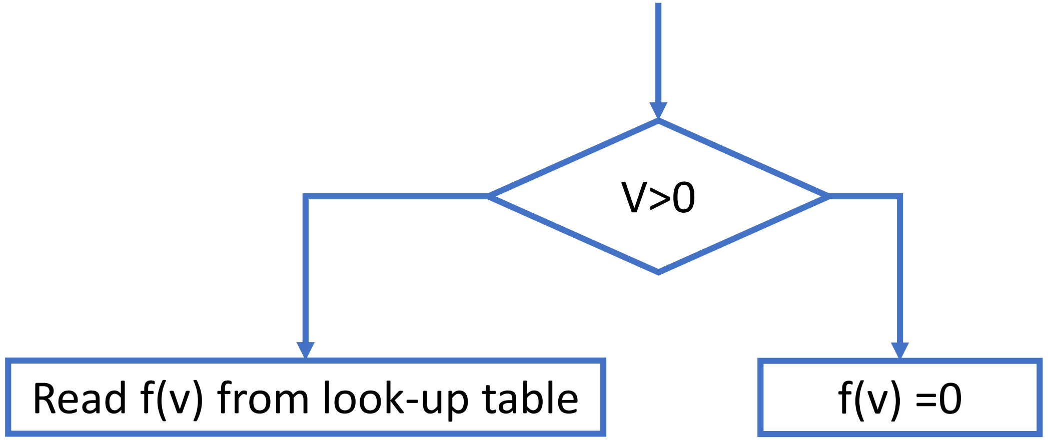
Artificial neuron



- Write a program for **an artificial neuron: y = f(v)**.

f(v) is a nonlinear function, as in the figure.
f(v) is 0, if v is negative.
Else, it is a trigonometric function.
Use a “look-up table” to calculate f(v).
// v is in R0. calculate f(v) and return the result in R0
// Look-up table starts at address **f.**
Fofv:

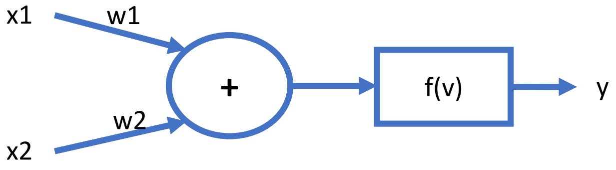




f:	f(0)
	f(1)
	f(2)
	f(3)
	…

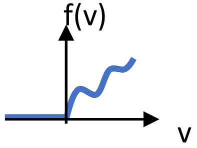
Look-up table

Artificial neuron

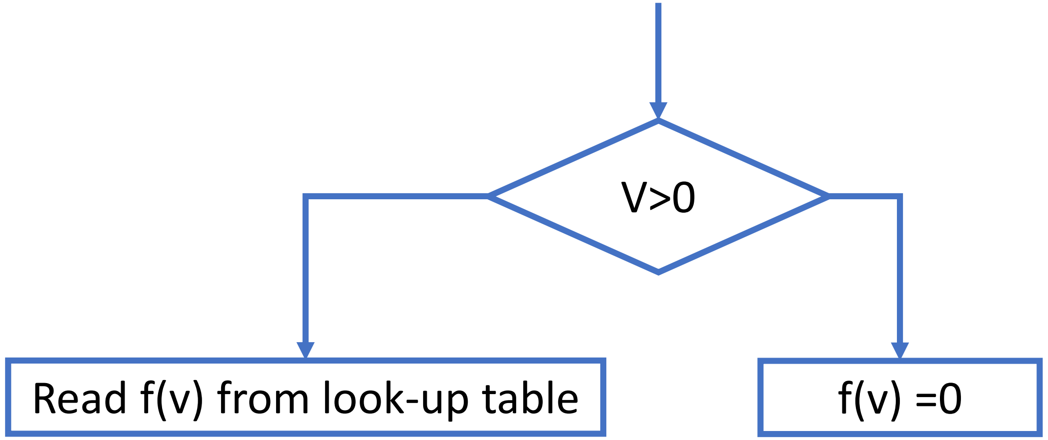


- Write a program for **an artificial neuron: y = f(v)**.

f(v) is a nonlinear function, as in the figure.
f(v) is 0, if v is negative.
Else, it is a trigonometric function.
Use a “look-up table” to calculate f(v).



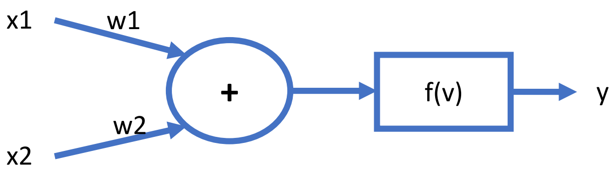

```
// v is in R0. calculate f(v) and return the result in R0
// Look-up table starts at address f.
Fofv:   LDR R3,=f
        CMP R0, #0
        MOVLE R0, #0
        LDRGT R0, [R3, R0, LSL #2]
```



f:	f(0)
	f(1)
	f(2)
	f(3)
	…

Look-up table

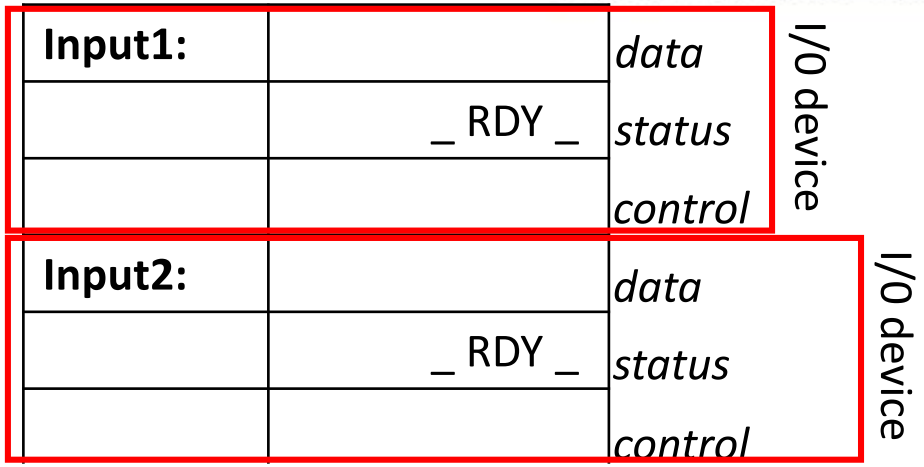
Artificial neuron

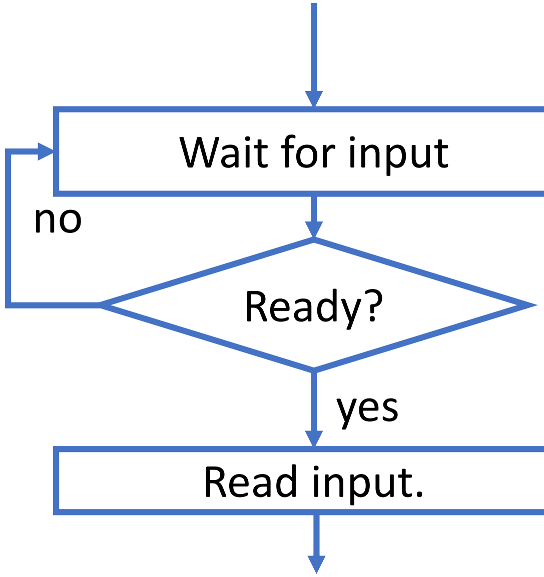


- Write a program for **an artificial neuron: y = f(v)**.

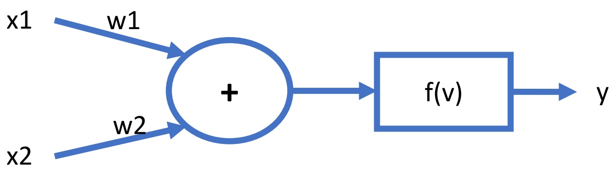
x1 and x2 are memory-mapped input, located at Input1 and Input2. Read the input data, when it is ready.

// Input1 and Input2 base addresses are in R5 and R6, resp.
// Write one subroutine that reads the I/O, whose base
//address is given with R0. Subroutine returns the data in R1.
Readinput:
Readloop:



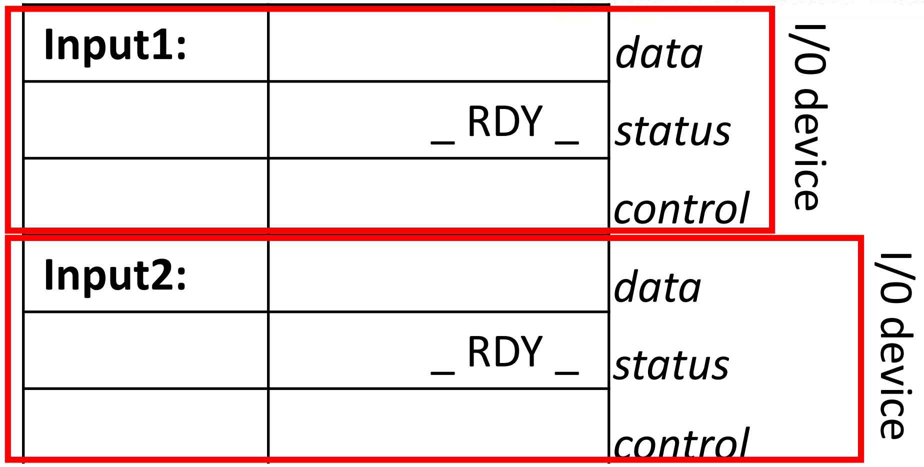


Artificial neuron

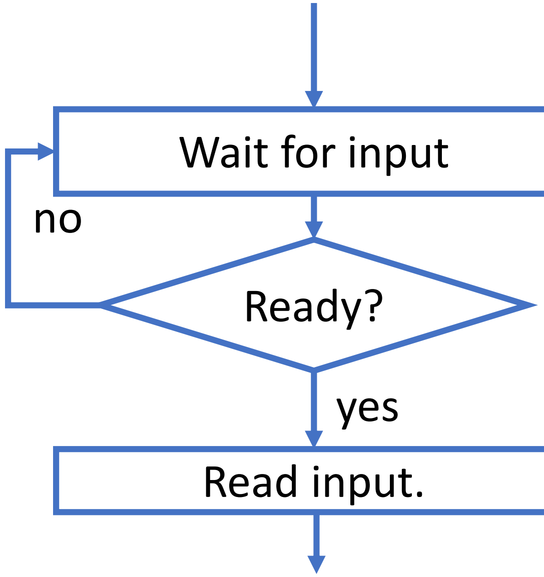


- Write a program for **an artificial neuron: y = f(v)**.

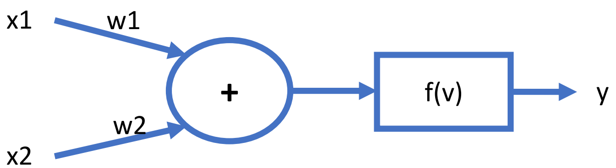
x1 and x2 are memory-mapped input, located at Input1 and Input2. Read the input data, when it is ready.




```
// Input1 and Input2 base addresses are in R5 and R6, resp.
// Write one subroutine that reads the I/O, whose base
//address is given with R0. Subroutine returns the data in R1.
Readinput:      PUSH {LR} // STMFD SP!, {LR}
Readloop:       LDR R1, [R0,#4]    // read status
          AND R1,R1,#2      // mask RDY bit
          CMP R1,#2
          BNE Readloop
          LDR R1,[R0]
          POP {PC} // LDMFD SP!, {PC}
          //return from subroutine
```



Artificial neuron



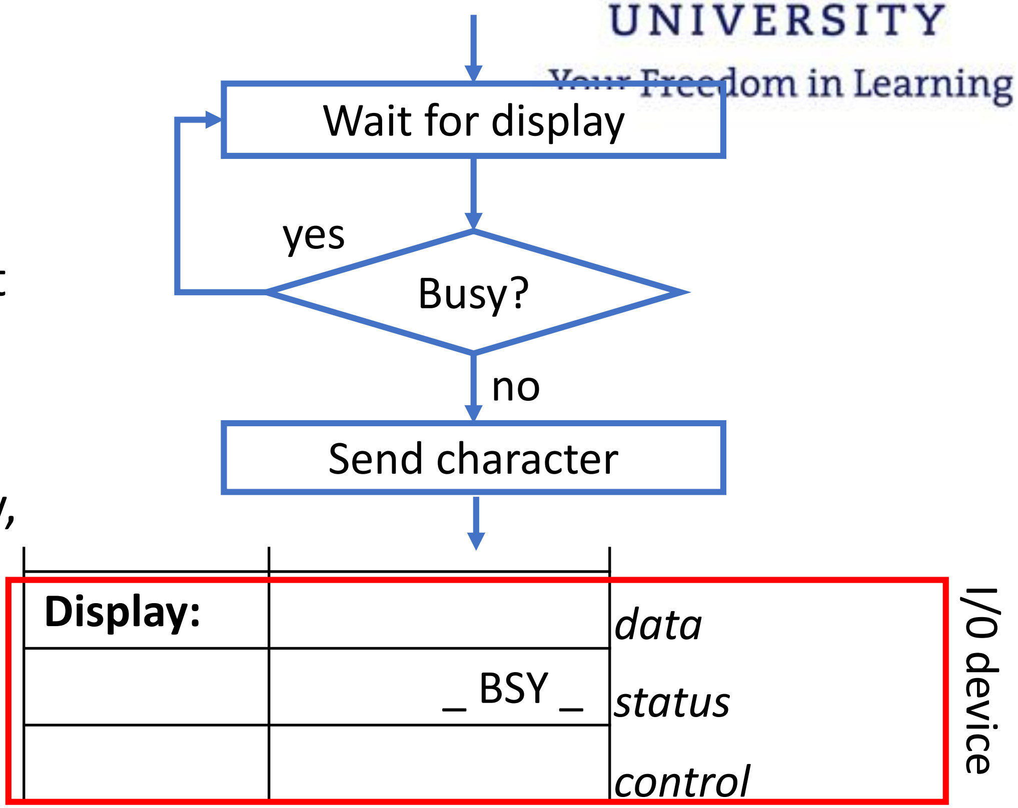
- Write a program for **an artificial neuron: y = f(v)**.

Calculate “y” and display the result, if the display device is not busy.

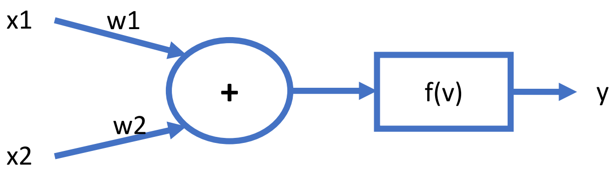
// Display base address is in R7. calculated f(v) is in R0.

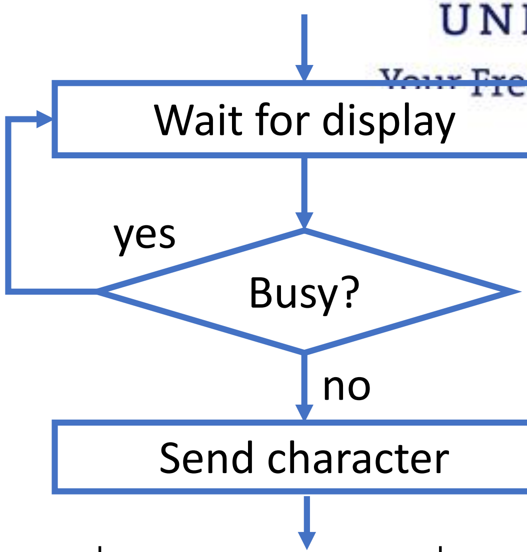
// If the BSY bit is 1, display is busy. Send R0 content to display,
//when display is not busy.

Waitdisplay:



Artificial neuron





Display:		*data*
	_ BSY _	*status*
		control

I/0 device

- Write a program for **an artificial neuron: y = f(v)**.

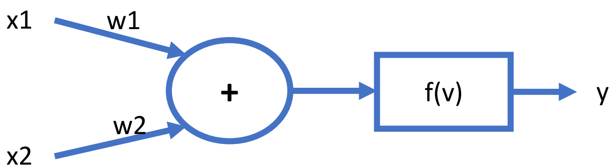
Calculate “y” and display the result, if the display device is not busy.

// Display base address is in R7. calculated f(v) is in R0.

// If the BSY bit is 1, display is busy. Send R0 content to display,
//when display is not busy.

```
Waitdisplay:    LDR R1,[R7,#4]    // read status
                AND R1,R1,#2      // mask BSY bit
                CMP R1,#2
                BEQ Waitdisplay
                STR R0,[R7]
```

Artificial neuron

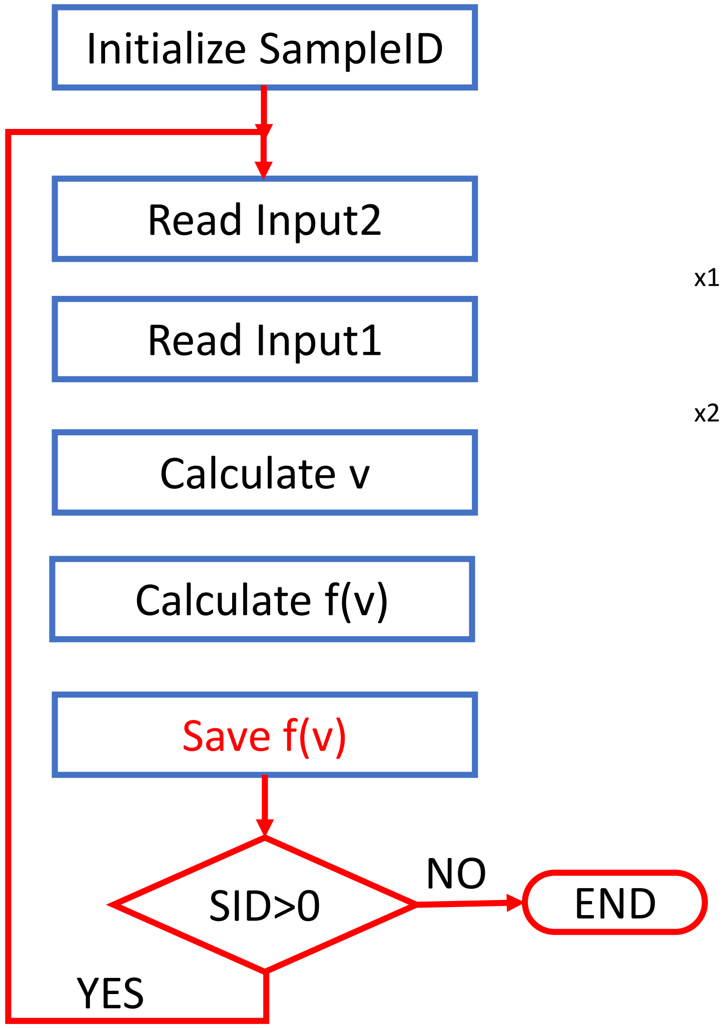


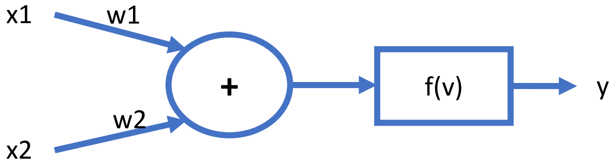
- Write a program for **an artificial neuron: y = f(v)**.

Read Input2	*R0 points the base address,Input2 (R6). Call subroutine Readinput R1 is x2, move it to R2*
Read Input1	*R0 points the base address, Input1 (R5). Call subroutine Readinput R1 is x1.*
Calculate v	*Calculate v: x1 and x2 are in R1 and R2. V is in R0*
Calculate f(v)	*Calculate f(v) v is in R0, result will return in R0*
Display f(v)	*Display base address is in R7. calculated f(v) is in R0.*

```
            LDR R5, =Input1
            LDR R6, =Input2
            LDR R7, =Display

            // Read input2 (x2) and input1 (x1)
            MOV R0, R6
            BL Readinput
            MOV R2, R1
            MOV R0, R5
            BL Readinput
Calcv:      MOV R1, R1, LSL #6          // 64.x1
            ADD R1, R1, LSL #1          // 64.x1 +  64.2.x1
            MVN R2, R2, LSL #1          // -2.x2-1
            ADD R2,R2,#1                // -2.x2
            ADD R0,R1,R2                // v

Fofv:       LDR R3,=f
            CMP R0, #0
            MOVLE R0, #0
            LDRGT R0, [R3, R0, LSL #2]

Waitdisplay: LDR R1,[R7,#4]             // read status
            AND R1,R1,#2                // mask BSY bit
            CMP R1,#2
            BEQ Waitdisplay
            STR R0,[R7]
END:        B END
Readinput:  STMFD SP!{LR}
Readloop:   LDR R1, [R0,#4]             // read status
            AND R1,R1,#2                // mask RDY bit
            CMP R1,#2
            BNE Readloop
            LDR R1,[R0]
            LDMFD SP!{PC}               //return from subroutine
```

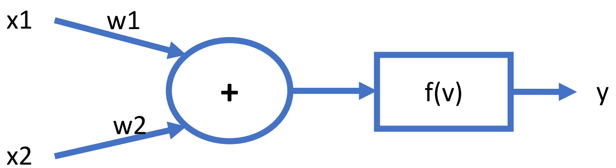
Artificial neuron – for an input set





SampleID	x1	x2	y
10	345	54098	?
9	8912	231	
8	5321	238	
7	7132	144	
6	43209	3	
…	…	…	
1	53	234	

Artificial neuron



- Write a program for **an artificial neuron: y = f(v)**.

Read Input2	*R0 points the base address,Input2 (R6).* *Call subroutine Readinput* *R1 is x2, move it to R2*
Read Input1	*R0 points the base address, Input1 (R5).* *Call subroutine Readinput* *R1 is x1.*
Calculate v	*Calculate v: x1 and x2 are in R1 and R2.* *V is in R0*
Calculate f(v)	*Calculate f(v)* *v is in R0, result will return in R0*
Save f(v)	*Save f(v) in stack*

```
                LDR R5, =Input1
                LDR R6, =Input2
                LDR R7, =Display

                // Read input2 (x2) and input1 (x1)
Loop:
                MOV R0, R6
                BL Readinput
                MOV R2, R1
                MOV R0, R5
                BL Readinput
Calcv:          MOV R1, R1, LSL #6          // 64.x1
                ADD R1, R1, LSL #1          // 64.x1 +  64.2.x1
                MVN R2, R2, LSL #1          // -2.x2-1
                ADD R2,R2,#1                // -2.x2
                ADD R0,R1,R2                // v

Fofv:           LDR R3,=f
                CMP R0, #0
                MOVLE R0, #0
                LDRGT R0, [R3, R0, LSL #2]

                PUSH R0                     // push result into stack


          END:  B END
Readinput:      STMFD SP!, {LR}
Readloop:       LDR R1, [R0,#4]             // read status
                AND R1,R1,#2 // mask RDY bit
                CMP R1,#2
                BNE Readloop
                LDR R1,[R0]
                LDMFD SP!, {PC}             //return from subroutine
```

MEF UNIVERSITY
Your Freedom in Learning

C to Assembly – For loop

- translate the following C code to assembly

.

```
for (i=0; i<8; i++) {
    a[i]=b[7-i];
}
```

C to Assembly – For loop

- translate the following C code to assembly

```
for (i=0; i<8; i++) {
   a[i]=b[7-i];
}
```

```
.global _start
_start:
        MOV r0, #7 // i
    LDR r1, =arrayb // load address of arrayb
    LDR r2, =arraya // a[i] starts here
 Loop:
    //RSB r3, r0, #7 // index = 7−i
     MVN r3,r0 // -i-1
     ADD r3,r3,#8 // -i-1+8=-i+7
    LDRB r5, [r1, r3] // load b[7−i]
    STRB r5, [r2, r0] // store into a[i]
    SUBS r0, r0, #1 // i——
    BGE Loop
end:     B end
arrayb: .word 0x0A090807,0x06050403
arraya: .byte 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0  // allocate some space
        .end
```