Department of Electrical and Computer Engineering

Howard University

Washington DC 20059

EECE 416 Microcomputer Design

Fall 2024

Design Project

By:

Juwon Wharwood

```
.equ swtch_ctrl, 0xff200040
.equ hex_ctrl, 0xff200020
.equ paint,0xc8000000
.equ write,0xc9000000
.global _start
_start:
    ldr r4,=swtch_ctrl
    ldr r5,=hex_ctrl
    ldr r11,=number_lookup
    mov r6,#0
    mov r7,#0
    mov r8,#0
    mov r9,#0
    mov r10,#0x01
    mov r12,#0

    bl store
    bl convert
```

To begin all relevant registers were determined and the first 4 registers were purposefully excluded.

The first objective is to store each digit of the number determined by the switches and separate them. The second objective is to convert it to a decimal number according to the guidelines of the project.

Number	SW3	SW2	SW1	SW0
1000	1	0	0	0
1101	1	1	0	1
0100	0	1	0	0

To achieve these results the store and convert functions were developed and utilized.

```
store:
    push {r10,r6,r4}
    mov r0,#0// thousand
    mov r1,#0//hundred
    mov r2,#0//tens
    ldr r3,=0//ones
    ldr r6,[r4]
    and r3, r6,r10
    lsr r6,r6,#1
    and r2,r6,r10
    lsr r6,r6,#1
    and r1,r6,r10
    lsr r6,r6,#1
    and r0,r6,r10
    pop {r10,r6,r4}
    bx lr
```

The store function separates each digit using the and function and puts the into a register that makes it easy for them to be returned to main.

```
convert:
    push {r7}
thousand:
    cmp r0,#1
    bne hundred
    add r7,r7,#1000
    b hundred
hundred:
    cmp r1,#1
    bne tens
    add r7,r7,#100
    b tens
tens:
    cmp r2,#1
    bne ones
    add r7,r7,#10
    b ones
ones:
    cmp r3,#1
    bne return
    add r7,r7,#1
    b return
return:
    mov r0,r7
    pop {r7}
    bx lr
```

The store function was then followed by the convert function where we determine our decimal number in terms of hundred thousands tens and ones with the right place value and we add them together and return them to main.

Each subsection is used to determine whether there is a one of a zero in that specific position and assigns a value accordingly. If there is a 1 in that position a value of 1,10,100 or 1000 will be added to the register storing the value.

The sum is moved to r0 in the return section of the function to prevent clobbering registers.

```
again:
    push {r0}
    cmp r0,#0
    blt stop
    bl division
    str r1,[r5]
    pop {r0}
```

In this section of main we push r0 into a stack to retain its original value. Afterwards we enable a stop condition to ensure our code stops when the counter reaches 0000. If this condition is not met we enter our division stage the result is then displayed on the hex display.

```
division:
    push {r7,r8,r9,r11,r12}
    mov r1,#0
    mov r7,#0
    mov r8,#0
    mov r9,#0
    mov r12,#0
Comp1000:
    cmp r0,#1000 //compares
    blt assign0_1000// if tl
    subs r0,r0,#1000// divi
    add r7,r7,#1
    cmp r0,#1000
    bge Comp1000// if the nu
    lsls r7,r7,#2// multipl
    ldr r8,[r11,r7]// using
    lsls r8,r8,#24// shift
    add r1,r1,r8// add to r
    b Comp100
assign0_1000://this assigns
    ldr r8,[r11]
    lsls r8,r8,#24
    add r1,r1,r8
    b Comp100// we jump to
```

```
Comp100:
    cmp r0,#100 //compa
    blt assign0_100// i
    subs r0,r0,#100// d
    add r9,r9,#1
    cmp r0,#100
    bge Comp100// if th
    lsls r9,r9,#2// mul
    ldr r8,[r11,r9]// u
    lsls r8,r8,#16// sh
    add r1,r1,r8// add
    b Comp10
assign0_100://this assi
    ldr r8,[r11]
    lsls r8,r8,#16
    add r1,r1,r8
    b Comp10// we jump
```

```
Comp10:
    cmp r0,#10//compa
    blt assign0_10//
    subs r0,r0,#10//
    add r12,r12,#1
    cmp r0,#10
    bge Comp10// if
    lsls r12,r12,#2/
    ldr r8,[r11,r12]
    lsls r8,r8,#8//
    add r1,r1,r8// a
    b sumcheck
assign0_10://this as
    ldr r8,[r11]
    lsls r8,r8,#8
    add r1,r1,r8
    b sumcheck// we
```

```
sumcheck:
    lsls r0,r0,#2// multiply
    ldr r8,[r11,r0]// using p
    add r1,r1,r8// add to r1
    pop {r7,r8,r9,r11,r12}
    bx lr
```

To ensure that there are no issues with registers having previous values we assign them to 0 before division begins. This division code is a retooled version of what was used for assignment 5 with a simple change to prevent instability issues of previous design and a new section included for the thousands calculations. The thousands, hundreds and tens subdivisions work as following. The counter is compared to required place value number, if the counter is less than that number we assign a zero to that place value location otherwise we follow our division steps by subtracting and storing the quotient until the number is less than that desired place value location. Dividing or assigning a zero to that desired place value we jump to a lower subdivision and repeat the process. The sumcheck section is different as the ones position does not require any division. In each section a lookup table is accessed using the quotient as an offset and the display information from the lookup table is combined to represent the correct number on the hex-display

```
    push {r0,r2,r6,r7,r8,r9,r10,r12}
    ldr r6,=vga_look
    ldr r7,=paint
    ldr r8,=write
    mov r10,#0//y
    ldr r2,=0x7098
    mov r12,#0
loopY:
    MOV r9, #0//x
loopX:
    BL onePixel
    CMP r9, #200
    ADD r9, #1
    BNE loopX
    CMP r10, #200
    ADD r10, #1
    BNE loopY
    MOV r10, #1
    MOV r9, #1
    BL number_writer
    bl delay
    pop {r0,r2,r6,r7,r8,r9,r10,r12}
    sub r0,r0,#1
    b again
```

This is another section in main where several values of originally defined registers are altered. To prevent various errors the original values of the registers are pumped into a stack and returned to after this section of the code is done. The code then jumps back to the again section to repeat the process.

Register 6 is now assigned the memory location for the 2nd lookup table which is accessed later in number_writer.

Register 7 is assigned to paint which is used to change the colour of the VGA display.

Register 8 is assigned to write which is used to write the numbers on the VGA display.

LoopY and its subdivision LoopX are used to chain the colour of the VGA display and create a box that is in this care purple. The loop and function onePixel were used exactly as they were in class time and obtained during in class practice.

The number_writer function is a combination of my division function and the oneChar function obtained during in class practice. Both functions combined allow for the writing of the number on VGA display.

The general coding design is located on the next page

The firs step is to separate all digits provided by switches

SW 3	SW 2	SW 1	SW 0
1	1	1	1

this gives F

objective is to isolate each one 3 note placevalue to get 1111_{10}

into a reg $r8$

0	0	0	0	1	1	1	1

AND it with 0x01

AND 0 0 0 0 0 1

=

0	0	0	0	0	0	0	1

store this in new reg

rotate right

0	0	0	0	0	1	1	1

and AND 3 more times store each in a new register

Now we have 4 registers that should have specific 1s or zeros in them

r0, r1, r2, r3
th h t o

cmp #1 bne
else r0 = 1000

cmp #1 bne
else r1 = 100

cmp #1 bne
r2 = 10

cmp #1 bne

total
r0+r1+r2+r3 = 1111_{10} → store in r0

See division on next page

Now we look at division

r0 → into fn

Check for thosands	check for hundreds	check for tens	ones
No thousands get zero from lookup table rotate 24 spots then	No hundreds? get zero from lookup table rotate 16 spots then	No tens get zero from lookup table rotate 8 spots then	No ones get zero from lookup table

	check for hundreds	check for tens	ones
Divide use quotient as offset for lookup table then rotate 24 spots here	Divide, use quotient as offset of lookup table, then rotate 16 spots	Divide, use quotient as offset for lookup table rotate 8 spots then	Use rem as offset for lookup table

add values together here

Lookup table illustration:

- word 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7C, 0x07, etc

Quotient Q is now offset

Eg Q = 7

7 x 4 = 28

· word	0x3f	0x06,	0x5b	0x4f	0x66	0x6d	0x7C	0x07	0x7f	0x67
	0	4	8	12	16	20	24	28	32	36

into new reg

rotate accordingly to the left

add accordingly eg 0x06 shifted 24 spots 0x06 00 00 00

∴ | | | | = | 06 | 06 | 06 | 06 |

Store this to display =

This needs to be in a loop r0 must be editted & several times

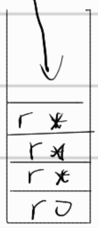
shove into a stack ro counter back to stack

do division

ro pop and then reduce ro at start of loop

VGA display DO THIS before reducing ro

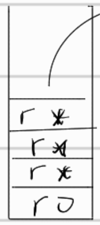
push back into a stack. Registers all used? push into stack



assign new values to registers

get prepared to paint & write

perform operations



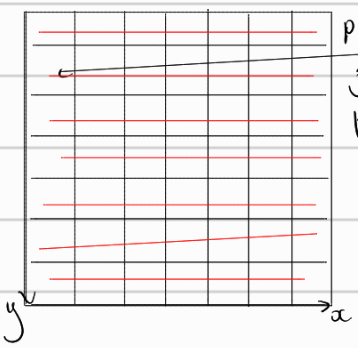
pop then reduce r0

and we start loop again

paint

Loop paint horizontal row then reset to zero

jump to new row that is one line below paint that row too for specific x & y ranges



paint this row then

jump to paint this row

for a range of x & y

After we paint now we write. Repeat division step with a twist insted we access new lookuptable

asciz. "0 1 2 3 4 5 6 7 8 9" each one is one byte

Quotient Q is not multiplied this time

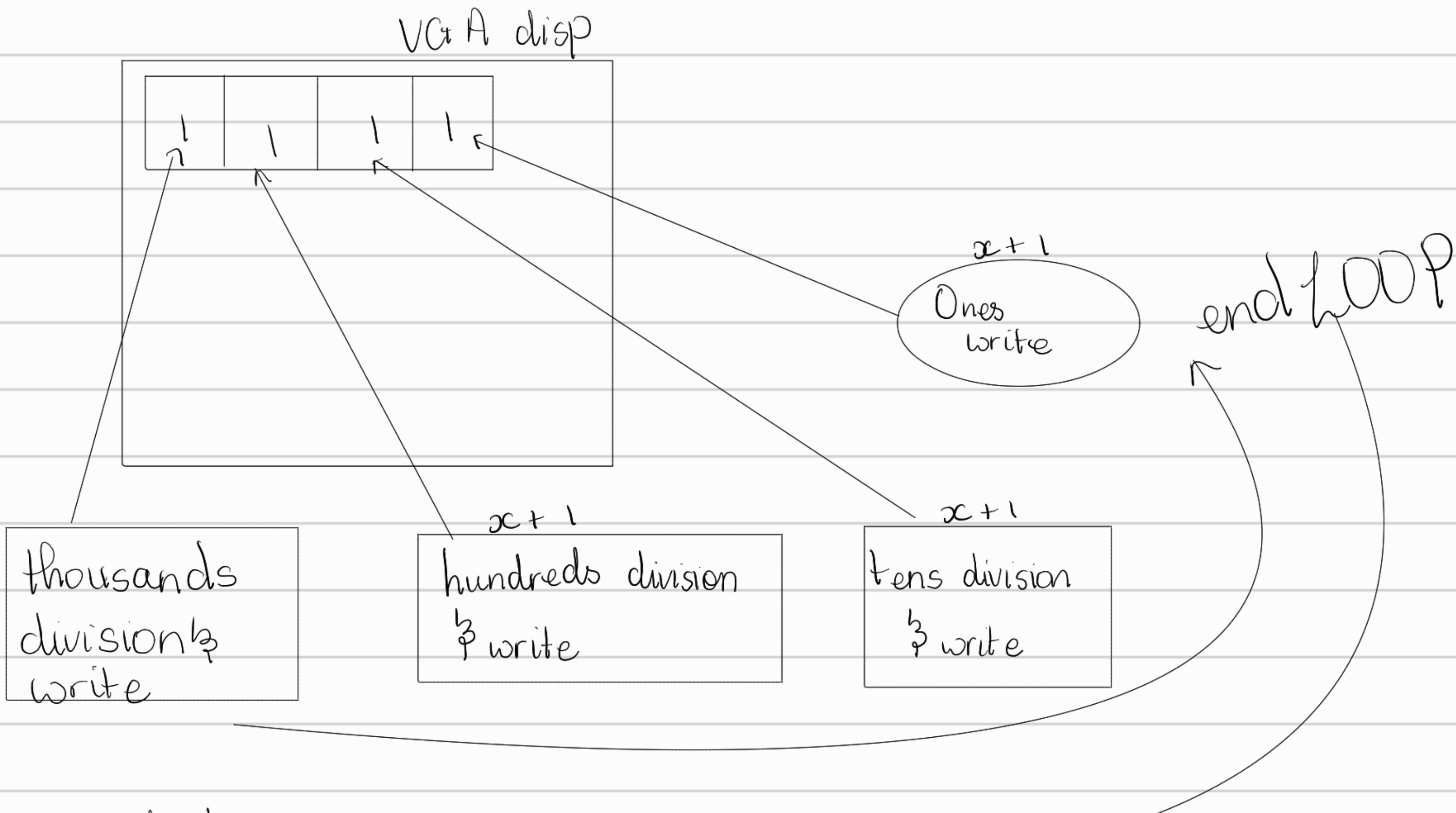
Q = 7

0	1	2	3	4	5	6	7	8	9

load to register

store to writing address for VGA

increment x for each division category



back to main code use a delay fn pop registers as intended
before reduce r0 (counter) do everything AGAIN