ECE/CSE 412

Lab #

Eugene Rockey, Copyright 2025

Report (85 Points)

Demo (15 Points)

PROJECT TITLE

*[Student Name]*

ECE 412

Abstract

*Briefly discuss the step-by-step procedures you took to perform this project (its experiments) and what hardware and software were involved if any. Basically discuss why you performed the project (purpose/goal/intent of learning), how you performed the project (step by step procedures), and what you used (equipment, tools, software, hardware, etc…), and finally state your results (successful or not/ was the goal achieved and how do you know it was achieved (data, tangible results, etc…))*

Body

*Mention your platform for performing this project, Windows, Linux, Unix, OS/X, etc… and if you had any problems using your platform – how did you solve the problems? This first lab is to be performed individually. Include detailed discussions about all the work you did in each part, the steps, the results, the data, etc… Include snapshots, block diagrams, charts, tables, equations, etc… at your discretionA graph with a line

AI-generated content may be incorrect.*

Source Code (Software)

*Neatly list all your original and/or modified assembly code involved in this project here with all fields and comments. Align all four Assembly columns.*

Sorting Algorithm Testing Code:

; sorting algorithm testing source code

;

; Created: 2/14/2025 12:01:11 AM

; Author : Joe Maloney

;

.MACRO U16\_CP ; args - rdH,rdL,rrH,rrL compares rdH:rdL to rrH:rdL

CP @1 , @3

CPC @0 , @2

.ENDMACRO

.MACRO U16\_ADD ; args rdH,rdL,rrH,rrL adds rdH:rdL to rrH:rrL and stores in rdH:rdL

ADD @1 , @3

ADC @0 , @2

.ENDMACRO

.MACRO U16\_SUB ; args rdH,rdL,rrH,rrL subtracts rrH:rrL from rdH:rdL and stores in rdH:rdL

SUB @1 , @3

SBC @0 , @2

.ENDMACRO

.MACRO U16\_PUSH ; args - rrH,rrL pushes uint onto stack

PUSH @1

PUSH @0

.ENDMACRO

.MACRO U16\_POP ; args - rrH,rrL pops uint from the stack

POP @0

POP @1

.ENDMACRO

; Replace with your application code

.LISTMAC

.EQU CHAR\_MAX=0xFF

.CSEG

.ORG 0x0

CLR r0 ; clear all registers prior to application start

CLR r1

CLR r2

CLR r3

CLR r4

CLR r5

CLR r6

CLR r7

CLR r8

CLR r9

CLR r10

CLR r11

CLR r12

CLR r13

CLR r14

CLR r15

CLR r16

CLR r17

CLR r18

CLR r19

CLR r20

CLR r21

CLR r22

CLR r23

CLR r24

CLR r25

CLR r26

CLR r27

CLR r28

CLR r29

CLR r30

CLR r31

USART\_Init: LDI r16 , 0x0 ; Set baud rate to UBRR0

STS UBRR0H, r16

LDI r16 , 103 ; 49 for 20K baud, 103 for 9600, 12 for 76800

STS UBRR0L, r16

LDI r16 , (1<<RXEN0)|(1<<TXEN0) ; enable reciver/transmitter

STS UCSR0B, r16

LDI r16 , (0<<USBS0)|(3<<UCSZ00) ; Set frame format: 8data, 1stop bit

STS UCSR0C, r16

start: RCALL next ; reserve first 2 bytes on stack for storing the test count

next: RCALL getTestCount ; get the number of tests to be performed from the uart

RCALL testLoop ; run the specified number of tests, getting a new dataset from the host PC each time

end: JMP end

qSortTest: RCALL getData ; load new dataset from host PC

LDI XL , 0x00

LDI XH , 0x01

LD r2 , X+

LD r3 , X+ ; set X pointer to array start address, and r3:r2 to array length for quicksort test

RCALL sendACK ; start timer on host PC

RCALL quickSort

RCALL testComplete ; stop timer on host PC

RET

quickSort: LDI r16 , 0x1

CLR r17 ; use r17:r16 for a constant uint 0x0001

U16\_CP r17 , r16 , r3 , r2 ; base case - break if length is 1 or 0

BRGE qSortR

RCALL part ; after partitioning, the ending address of the pivot is stored in the Y pointer

U16\_PUSH YH , YL ; store pivot location on stack

U16\_SUB YH , YL , XH , XL ; calculate lower array size in bytes

LSR YH ; This number is guaranteed to be even

ROR YL ; divide by 2 to get number of elements, ror is lsr w/ carry bit

U16\_SUB r3 , r2 , YH , YL ; calculate number of elements in upper half, including pivot

U16\_SUB r3 , r2 , r17 , r16 ; Y=Y-1, get rid of the pivot

U16\_PUSH r3 , r2 ; store upper array size on stack

MOV r3 , YH

MOV r2 , YL ; move array length into r3:r2 for next call to quicksort

RCALL quickSort ; lower half, X is equivalent for this call, r3:r2 holds new length

U16\_POP r3 , r2 ; move upper array length into r3:r2 for next call to quicksort

U16\_POP XH , XL ; restore X pointer to previous pivot

U16\_ADD XH , XL , r17 , r16

U16\_ADD XH , XL , r17 , r16 ; move Y to element after previous pivot (lower byte of first element in upper half array)

RCALL quickSort ; Upper half, X is at the element after previous pivot, r3:r2 holds upper half length

qSortR: RET

; Array start address is X (array start and pivot are the same element), array length stored at r3:r2

part: MOV r0 , XL

MOV r1 , XH ; write down pivot address in r1:r0

MOV YL , XL

MOV YH , XH ; set y pointer to first (non-pivot) value in array

LD r4 , X+

LD r5 , X+ ; load the pivot into r5:r4

CLR r17

LDI r16 , 0x1 ; use r17:r16 to increment loop counter

CLR r7

MOV r6 , r16 ; use r7:r6 for loop counter, start at 1

partL1: U16\_CP r7 , r6 , r3 , r2 ; stop loop when counter == r3:r2 (array length)

BREQ partR

LD r12 , X+

LD r13 , X+ ; load the current value into r13:r12

U16\_CP r13 , r12 , r5 , r4 ; compare value to pivot

BRLO partL2

JMP partL3 ; don't swap if value>pivot

partL2: RCALL qSwap ; swap the pivot and value if value is less than pivot

partL3: U16\_ADD r7 , r6 , r17 , r16 ; increment loop counter

JMP partL1

partR: RCALL qSwapPivot ; swap \*Y and pivot

RET

qSwapPivot: LD r14 , Y+

LD r15 , Y+ ; store value to be swapped in r15:r14

U16\_SUB YH , YL , r17 , r16

U16\_SUB YH , YL , r17 , r16 ; send Y pointer back to address of value to be swapped

MOV XL , r0

MOV XH , r1 ; restore X to pivot location

ST X+ , r14

ST X+ , r15 ; store \*Y at original pivot location

ST Y+ , r4

ST Y+ , r5 ; store pivot at address of Y

U16\_SUB YH , YL , r17 , r16

U16\_SUB YH , YL , r17 , r16 ; send Y pointer back to address of value to be swapped

U16\_SUB XH , XL , r17 , r16

U16\_SUB XH , XL , r17 , r16 ; send X pointer back to original array start addr, used for calculating upper/lower half length in qsort

RET

; swaps values at (Y+1) and X, does not change X, Y=(Y+1)

qSwap: LD r15 , -X

LD r15 , -X ; retract X back to the address of the value to be swapped

U16\_ADD YH , YL , r17 , r16

U16\_ADD YH , YL , r17 , r16 ; increment Y pointer

LD r14 , Y+

LD r15 , Y+ ; load value to be swapped from Y pointer into r15:r14

U16\_SUB YH , YL , r17 , r16

U16\_SUB YH , YL , r17 , r16 ; send Y pointer back to address of value to be swapped

LD r18 , X+

LD r19 , X+ ; load other value to be swapped into r19:r18

U16\_SUB XH , XL , r17 , r16

U16\_SUB XH , XL , r17 , r16 ; decrement X pointer to original location

ST X+ , r14

ST X+ , r15 ; store the \*(Y+1) value at the original location of X

ST Y+ , r18

ST Y+ , r19 ; store the \*X value at (Y+1)

U16\_SUB YH , YL , r17 , r16

U16\_SUB YH , YL , r17 , r16 ; Y now addresses (Y+1) from the original Y

RET

getTestCount:RCALL sendACK

LDI XL , 0xFE

LDI XH , 0x08 ; set X to last SRAM location

RCALL uint16\_Rx ; get and store test count in last SRAM location

RET

testLoop: LDI XL , 0xFE

LDI XH , 0x08 ; set X pointer to the number of tests to run

CLR r17

LDI r16 , 0x1 ; use r17:r16 to increment loop counter

LD r24 , X+

LD r25 , X ; load test count into r25:r24, use for loop stop condition

CLR r23

CLR r22 ; use r23:r22 for loop counter

testL1: U16\_CP r23 , r22 , r25 , r24

BREQ testR

RCALL bSortTest ; change this call from bSortTest/qSortTest

CLR r17

LDI r16 , 0x1 ; use r17:r16 to increment loop counter

U16\_ADD r23 , r22 , r17 , r16 ; increment loop counter

JMP testL1

testR: RET

; uses X and Y for indirection to data, Z for accumulator

bSortTest: RCALL getData

RCALL sendACK

RCALL bubbleSort

RCALL testComplete

RET

bubbleSort: LDI XL , 0x0 ; set X to location of n

LDI XH , 0x1

LDI YL , 0x4

LDI YH , 0x1 ; set Y to second data location

CLR ZL

CLR ZH ; set Z to 0

LD r0 , X+

LD r1 , X+ ; store the number of numbers (n) in r1:r0,X now points at low byte of first uint16

MOV r18 , r0

MOV r19 , r1 ; use r19:r18 for outer loop end condition check

LD r2 , -X

LD r2 , -X ; decrement X to addr of last data uint low byte

U16\_ADD XH , XL , r1 , r0 ; add n to X address, doing this twice because uint16 is 2 bytes large

U16\_ADD XH , XL , r1 , r0 ; add n to X address, this makes X point to the low byte of 1 of the last uint16

MOV r0 , XL

MOV r1 , XH ; load the end of data address into r1:r0, this is stop condition for the loops

LDI XL , 0x2

LDI XH , 0x1 ; X points to first data uint16 low byte

CLR r2

CLR r3

CLR r4

CLR r5

CLR r17

LDI r16 , 0x1 ; use r17:r16 to decrement the loop stop condition

U16\_SUB r19 , r18 , r17 , r16 ; outer loop runs (n-1) times

CLR r6

CLR r7 ; use r7:r6 for outer loop iterator

bubbleL1: U16\_CP r7 , r6 , r19 , r18 ; outer loop r7:r6 is iterator, starts at 0, r19:r18 is stop condition, breaks at i = (n-1)

BREQ bubbleR ; stop sorting

bubbleL2: U16\_CP r1 , r0 , XH , XL

BREQ bubbleL2end

MOV ZL , XL

MOV ZH , XH ; Z reg used for swap, needs to point to original location of first uint low byte

LD r2 , X+

LD r3 , X+ ; Load first uint16 into r3:r2

LD r4 , Y+

LD r5 , Y+ ; Load second uint16 into r5:r4

U16\_CP r3 , r2 , r5 , r4 ; compare the numbers

BRSH callSwap ; swap if \*X >= \*Y, brsh is breq for unsigned numbers

JMP bubbleL2

callSwap: RCALL bubbleSwap ; swap the numbers if number at X >= number at Y

JMP bubbleL2

bubbleL2end:LDI XL , 0x2

LDI XH , 0x1 ; reset the X pointer to first uint low byte

LDI YH , 0x1

LDI YL , 0x4 ; reset the Y pointer to first uint low byte

U16\_ADD r7 , r6 , r17 , r16 ; increment loop counter

U16\_SUB r1 , r0 , r17 , r16

U16\_SUB r1 , r0 , r17 , r16 ; decrement the inner loop stop condition address by 2 bytes, skip the last element that was sorted in the next iteration

JMP bubbleL1

bubbleR: RET

; working regs r21:r20, swaps uint16, assumes \*Z is low byte of first uint,r3:r2 is first uint,r5:r4 is second uint

bubbleSwap: MOV r20 , r2

MOV r21 , r3 ; store first uint in r17:r16

MOV r3 , r5

MOV r2 , r4 ; write second uint into first uint's registers

ST Z+ , r2

ST Z+ , r3 ; write second uint into first's sram location

ST Z+ , r20

ST Z+ , r21 ; write first uint into second's sram location

RET

getData: LDI r26 , 0x00 ; set X to start of sram

LDI r27 , 0x1

RCALL sendACK

RCALL uint16\_Rx ; get first uint16 at 0x100, this is the number of numbers (n) in the dataset

LDS r0 , 0x100

LDS r1 , 0x101 ; load n into r0,r1.

CLR ZL

CLR ZH ; use Z for accumulator, and r1:r0 for compare

getDataL1: U16\_CP r1 , r0 , ZH , ZL

BREQ getDataR

RCALL uint16\_Rx ; get the next dataset number

ADIW ZL , 1

JMP getDataL1

getDataR: RET

uint16\_Rx: RCALL USART\_Rx ; receives a single byte from

ST X+ , r17

RCALL USART\_Rx

ST X+ , r17

RET

uint16\_Tx: LD r16 , X+

RCALL USART\_Tx

LD r16 , X+

RCALL USART\_Tx

RET

testComplete:LDI r16 , 0xFF

RCALL USART\_Tx

RET

sendACK: LDI r16 , 0xF0

RCALL USART\_Tx

RET

; Wait for empty transmit buffer

USART\_Tx: LDS r17 , UCSR0A ; working: r17, sends byte in r16 , read uart status reg

SBRS r17 , UDRE0 ; infinite loop until I/0 is empty, checks if data empty bit is set in uart status reg

RJMP USART\_Tx

; Put data (r16) into buffer, sends the data

STS UDR0 , r16

RET

USART\_Rx: LDS r17 , UCSR0A ; reads uart sreg into r17, blocking the read of the uart data register until data ready

SBRS r17 , RXC0

RJMP USART\_Rx

LDS r17 , UDR0 ; read uart data register into r17

RET

; table - a non uart dataset used for debugging w/ the simulator

table: .DB 0x64 , 0x0 , 0xc3 , 0xca , 0x38 , 0xad , 0xbc , 0x79 , 0xfc , 0x8e , 0x3a , 0xbd , 0x53 , 0x83 , 0x69 , 0xcb , 0x67 , 0x63 , 0x55 , 0xc4 , 0x09 , 0xc0 , 0xc5 , 0x5a , 0xd3 , 0x01 , 0xc0 , 0x40 , 0x36 , 0x3f , 0x9d , 0xea , 0xf8 , 0x9e , 0x9c , 0xea , 0x15 , 0x51 , 0x07 , 0xfe , 0x58 , 0xee , 0x66 , 0xca , 0xec , 0x9a , 0x12 , 0x3e , 0x0d , 0xf6 , 0xa2 , 0x7b , 0xe6 , 0x0b , 0x93 , 0x2f , 0x78 , 0x24 , 0x4c , 0x9a , 0xf7 , 0x81 , 0x04 , 0x90 , 0x71 , 0x3e , 0xf5 , 0xa8 , 0xbd , 0xbe , 0x09 , 0x1c , 0xfb , 0xfd , 0xd5 , 0x4a , 0x89 , 0x24 , 0xfd , 0x27 , 0x00 , 0xa1 , 0x53 , 0x34 , 0xd6 , 0xec , 0xd7 , 0x60 , 0xfd , 0xc1 , 0x11 , 0x5d , 0x55 , 0x77 , 0x0c , 0x0d , 0xbc , 0x51 , 0xbb , 0x78 , 0x01 , 0x39 , 0x35 , 0xe4 , 0x5a , 0x82 , 0xae , 0xd9 , 0x92 , 0x74 , 0xea , 0x5f , 0x92 , 0x2d , 0x5a , 0x96 , 0xd1 , 0xbb , 0xc6 , 0x4b , 0x41 , 0x2e , 0xba , 0xb6 , 0xfc , 0x21 , 0x85 , 0xf8 , 0xa1 , 0x6a , 0xee , 0x5f , 0x6b , 0xdb , 0x2a , 0x75 , 0x33 , 0x71 , 0x6d , 0xe2 , 0x82 , 0xf4 , 0xee , 0x97 , 0x09 , 0x51 , 0xd7 , 0x57 , 0x0e , 0xfe , 0x75 , 0xd6 , 0xb6 , 0xaf , 0xda , 0x13 , 0xba , 0x4d , 0x00 , 0x27 , 0xeb , 0xe9 , 0x7d , 0x7b , 0x31 , 0x5b , 0x11 , 0x3d , 0xf2 , 0x8c , 0x2e , 0xef , 0x37 , 0x8a , 0xc7 , 0xf7 , 0x25 , 0xf4 , 0xd3 , 0xee , 0x82 , 0x64 , 0x8f , 0xb0 , 0x3d , 0xd6 , 0x85 , 0x22 , 0x9e , 0x3e , 0x67 , 0x2b , 0x36 , 0x9a , 0xd0 , 0x88 , 0x9e , 0xbf , 0x81 , 0x78 , 0x43 , 0x3b

getDataDebug:LDI XL , 0x0 ; same as getData, but reads from program flash instead of receiving data via uart

LDI XH , 0x1

LDI ZL , low(table\*2)

LDI ZH , high(table\*2) ; set Z to starting address of table

RCALL getuint16Debug ; get the number of uint data elements in table (n)

LDS r0 , 0x100

LDS r1 , 0x101 ; load n into r0,r1.

CLR YL

CLR YH ; use Y for accumulator, and r1:r0 for compare

debugL1: U16\_CP r1 , r0 , YH , YL

BREQ getDataDebugR

RCALL getuint16Debug ; get the next dataset number

ADIW YL , 1

JMP debugL1

getDataDebugR:RET

getuint16Debug:LPM r16 , Z+ ; same as getuint16, but loads uint16 from program memory for debugging

ST X+ , r16 ; store n low byte

LPM r16 , Z+

ST X+ , r16 ; store n high byte

RET

Schematics (Hardware)

*Include schematic(s) of circuits relevant to the project.*

*If there are none then type ‘none’*

Analysis

*Overall, discuss what you learned, how your learned it, and why you learned the information in this project in order to answer what, how, and why this information can be applied to real world embedded systems (give specific examples). Use your results/data. Do not refer to yourself.*

Conclusion

*Restate the purpose/goal of the project and state any conclusions you have based on the results and/or data obtained.*

References

*Cite all sources you researched and/or used to perform this lab*

*If no references were used then type ‘none’*