

Lab Report

LAB 13_STACK

I/- OBJECTIVE:

Understand the usage and operation of Stack and Stack Pointer (LIFO)
Using PUSH & POP instructions.

II/- TOOL – Software :

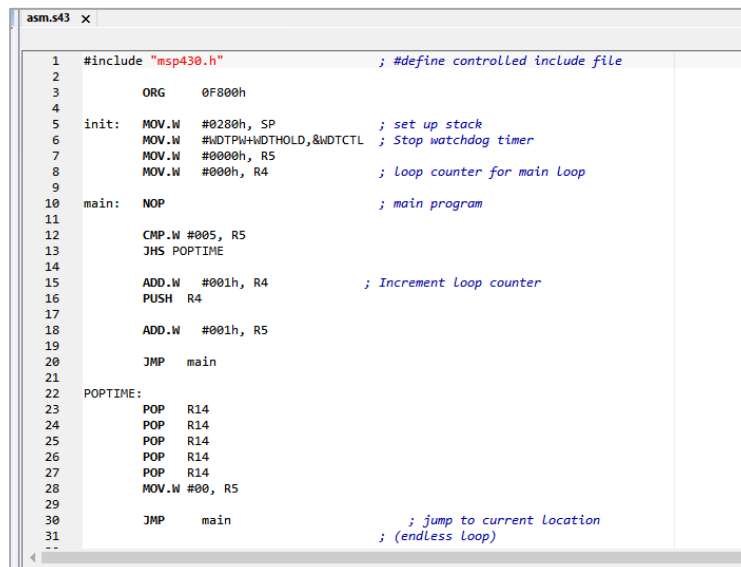
Download Attached Files : [Lab 13 \(Stack\).zip](#) (20.688 KB)

From : [Lab 13 \(Stack\)](#) in Blackboard .

Open file Stack type IAR IDE Workspace

Edit file asm.s43

 The window of file asm.s43 and lines of Instruction.



```
1  #include "msp430.h"          ; #define controlled include file
2
3      ORG      0F800h
4
5  init: MOV.W  #0280h, SP      ; set up stack
6        MOV.W  #WDTPW+WDTHOLD,&WDCTL ; Stop watchdog timer
7        MOV.W  #0000h, R5
8        MOV.W  #000h, R4      ; Loop counter for main loop
9
10 main:  NOP                  ; main program
11
12        CMP.W  #005, R5
13        JHS POPTIME
14
15        ADD.W  #001h, R4      ; Increment loop counter
16        PUSH  R4
17
18        ADD.W  #001h, R5
19
20        JMP   main
21
22 POPTIME:
23        POP   R14
24        POP   R14
25        POP   R14
26        POP   R14
27        POP   R14
28        MOV.W #00, R5
29
30        JMP   main          ; jump to current location
31                          ; (endless loop)
```



III/- PRACTICE

1)- Start simulating the program by clicking the "Download and Debug" button on the toolbar.

- Using single step through the loop one time. This is accomplished by clicking on the "Step Into" button or pressing F11

2)- Single step through assembly code in the program and observations Include what happens to the SP and the stack itself in memory with each stack related operation.

3)- Try changing the SP to a value 40 bytes lower using the watch window after executing *line 28* to simulate a new "frame" of operation.

a)- Include what happens to the data in the original stack frame.

b)- Could we use the memory area on the stack between the original stack frame and the new stack frame for temporary variables?

✚ The Screen of asm.s43 file in Stack operations, and Push and Pop Instruction .

The screenshot displays the Keil uVision IDE with the assembly file `asm.s43` open. The source code window on the left shows the following assembly instructions:

```
1 #include "map430.h" ; Define controlled include file
2
3 CBG 0FF00h
4
5 init: MOV.W #0200h, SP ; set up stack
6 MOV.W #0000h, WDTCTL ; Stop watchdog timer
7 MOV.W #0000h, R5
8 MOV.W #0000h, R4 ; loop counter for main loop
9
10 main: BCF ; main program
11
12 CMP.W #005, R5
13 JBS PCPTIME
14
15 ADD.W #001h, R4 ; Increment loop counter
16 PUSH R4
17
18 ADD.W #001h, R5
19
20 JMP main
21
22 PCPTIME:
23 POP R4
24 POP R4
25 POP R4
26 POP R4
27 POP R4
28 MOV.W #00, R5
29
30 JMP main ; jump to current location
31 ; (endless loop)
32
33
34
35
36
37
38
39 CBG 0FF00h
40 DC14 init ; set reset vector to "init" label
41 END
```

The register window in the center shows the current state of CPU registers. The SP (Stack Pointer) is at 0x0048. The register window also shows the current state of other registers, including R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, and the CYCLES counter.

The disassembly window on the right shows the corresponding machine code instructions. The instructions are listed with their addresses and the disassembled code. The instructions are:

- 00F000: MOV.W #0200h, SP ; set up stack
- 00F004: MOV.W #0000h, WDTCTL ; Stop watchdog timer
- 00F008: MOV.W #0000h, R5
- 00F00C: MOV.W #0000h, R4 ; loop counter for main loop
- 00F010: BCF ; main program
- 00F014: CMP.W #005, R5
- 00F018: JBS PCPTIME
- 00F01C: ADD.W #001h, R4 ; Increment loop counter
- 00F020: PUSH R4
- 00F024: ADD.W #001h, R5
- 00F028: JMP main
- 00F02C: CBG 0FF00h
- 00F030: DC14 init ; set reset vector to "init" label
- 00F034: END

The bottom status bar indicates the current address is 0x0200.



IV/- RESULT

2)- After loop 1 : We see

- The Program Counter (PC) is incremented (+2) after execute each Instruction .
(from 0xF804 to 0xf81C)
- The Status Register (SR) decreased (-2) from 0x0280 to 0x027E by Push R4 instruction (line 16)
- The Register R4 increment (+1) from 0x0000 to 0x0001 by ADD.W #001h, R4 instruction (line 15)
- The Register R5 increment (+1) from 0x0000 to 0x0001 by ADD.W #001h, R5 instruction (line 18)

After loop 5:

- The Register R5 increment (+5) and reached 0x005.
- Thus, CMP.W #005, R5 is the same dst = src (line 12)

=> JHS POPTIME Instruction will execute jump to poptime: (line 13)

- The Program Counter (PC) continuous incremented (+2) from 0xF820 to 0xf82A
- The Stack Pointer (SP) is incremented (+2) when run though each POP instruction,
from 0x0278 to 0x0280.
- The Register R5 = 0x00 by ADD.W #00h, R5 instruction (line 28)
- The Register R14 in decreased from 0x0005 to 0x0001 (after executed 5 POP instructions)

3)- When changing the SP to a value 40 bytes lower we have value of SP = 0x0240.

And after executing line 28 to simulate a new "frame" of operation, we have :

MOV.W #00, R5 (line 28)

- What happens to the data in the original stack frame?
- Yes, we could use the memory in the region between new stack frame (0x0240) and original stack farm (0x0280) for temporary variables.

Since the SP no longer points to this area, and as long as no other parts of the program attempt to use the stack in this area, we can safely store temporary variables here.

However, doing so could lead to stack management issues and potential bugs if the SP is reset or if any interruptions occur, which could overwrite or disrupt data in this region.

Loop 1								
Line	Instruction	PC	SP	SR	R4	R5	R14	Note
1	# include "msp430.h"							; # define controlled include file
2								
3	ORG 0F800h							
4								
5	init : MOV.W # 0280h, SP	0xF804	0x0280					; Set up Stack
6	MOV.W # WDPTW+WDTHOLD, &WDCTL	0xF80A						; Stop Watchdog timer
7	MOV.W # 0000h, R5	0xF80C				0x0000		
8	MOV.W # 0000h, R4	0xF80E			0x0000			; Loop counter for main loop
9								
10	Main : NOP	0xF810						; Main program
11								
12	CMP.W #005, R5	0xF814		0x0004				; (N = 1)
13	JHS POPTIME	0xF816						
14								
15	ADD.W #001h, R4	0xF818		0x0000	0x0001			; Increment Loop counter
16	PUSH R4	0xF81A	0x027E					
17								
18	ADD.W #001h, R5	0xF81C				0x0001		
19								
20	JMP main							
21	; After each loop then (Add.W #001h, R5) the value of R5 will incremen more +1. So After 5 main loop then R5 = 0x0005 , and JHS POPTIME will execute instruction CMP.W #005, R5 (Jump if higher or the same) and jumo to POPTIME							
22	POPTIME:							
23	POP R14	0xF820	0x0278				0x0005	
24	POP R14	0xF822	0x027A				0x0004	
25	POP R14	0xF824	0x027C				0x0003	
26	POP R14	0xF826	0x027E				0x0002	
27	POP R14	0xF828	0x0280				0x0001	
28	MOV.W #00, R5	0xF82A				0x0000		
29								
30	JMP main							; Jump to current location
31								; (enless loop)



V/- CONCLUSION

From each step in the program we observe: they work in accordance with the theory of instruction.

Single step through the program and submit your observations in a lab report.

- Include what happens to the SP and the stack itself in memory with each stack related operation.

Try changing the SP to a value 40 bytes lower using the watch window after executing line 28 to simulate a new "frame" of operation and submit your observations in a lab report.

- Include what happens to the data in the original stack frame.

[illegible]

Page 5 of 5

ms430 - WinCCSoftWare Management Tool - Project1 / 4.1.1

File Edit View Project Debug Simulator Tools Window Help

ms430 x

Registers 1

Name	Value	Access
PC	0x010	Read/Write
SP	0x0248	Read/Write
SR	0x0000	Read/Write
Reserved	0x0	Read/Write
V	0	Read/Write
SCG1	0	Read/Write
SCG0	0	Read/Write
Oncoff	0	Read/Write
CPWOff	0	Read/Write
GIE	0	Read/Write
N	0	Read/Write
Z	0	Read/Write
C	0	Read/Write
R4	0x001e	Read/Write
R5	0x0000	Read/Write
R6	0x0000	Read/Write
R7	0x0000	Read/Write
R8	0x0000	Read/Write
R9	0x0000	Read/Write
R10	0x0000	Read/Write
R11	0x0000	Read/Write
R12	0x0000	Read/Write
R13	0x0000	Read/Write
R14	0x001e	Read/Write
R15	0x0000	Read/Write
CYCLECOUNTER	486	Read/Write
CCTIMER1	486	Read/Write
CCTIMER2	486	Read/Write
CCSTEP	3	Read/Write

Disassembly

Address	OpCode	Comment
00F800	MOV.W #0280, SP	: set up stack
00F801	init	
00F802	MOV.W #0280, SP	
00F803	MOV.W #0280, SP	
00F804	MOV.W #0280, SP	
00F805	MOV.W #0280, SP	
00F806	MOV.W #0280, SP	
00F807	MOV.W #0280, SP	
00F808	MOV.W #0280, SP	
00F809	MOV.W #0280, SP	
00F80A	MOV.W #0280, SP	
00F80B	MOV.W #0280, SP	
00F80C	MOV.W #0280, SP	
00F80D	MOV.W #0280, SP	
00F80E	MOV.W #0280, SP	
00F80F	MOV.W #0280, SP	
00F810	MOV.W #0280, SP	
00F811	MOV.W #0280, SP	
00F812	MOV.W #0280, SP	
00F813	MOV.W #0280, SP	
00F814	MOV.W #0280, SP	
00F815	MOV.W #0280, SP	
00F816	MOV.W #0280, SP	
00F817	MOV.W #0280, SP	
00F818	MOV.W #0280, SP	
00F819	MOV.W #0280, SP	
00F81A	MOV.W #0280, SP	
00F81B	MOV.W #0280, SP	
00F81C	MOV.W #0280, SP	
00F81D	MOV.W #0280, SP	
00F81E	MOV.W #0280, SP	
00F81F	MOV.W #0280, SP	
00F820	MOV.W #0280, SP	
00F821	MOV.W #0280, SP	
00F822	MOV.W #0280, SP	
00F823	MOV.W #0280, SP	
00F824	MOV.W #0280, SP	
00F825	MOV.W #0280, SP	
00F826	MOV.W #0280, SP	
00F827	MOV.W #0280, SP	
00F828	MOV.W #0280, SP	
00F829	MOV.W #0280, SP	
00F82A	MOV.W #0280, SP	
00F82B	MOV.W #0280, SP	
00F82C	MOV.W #0280, SP	
00F82D	MOV.W #0280, SP	
00F82E	MOV.W #0280, SP	

Memory

Address	OpCode	Comment
00F800	MOV.W #0280, SP	
00F801	init	
00F802	MOV.W #0280, SP	
00F803	MOV.W #0280, SP	
00F804	MOV.W #0280, SP	
00F805	MOV.W #0280, SP	
00F806	MOV.W #0280, SP	
00F807	MOV.W #0280, SP	
00F808	MOV.W #0280, SP	
00F809	MOV.W #0280, SP	
00F80A	MOV.W #0280, SP	
00F80B	MOV.W #0280, SP	
00F80C	MOV.W #0280, SP	
00F80D	MOV.W #0280, SP	
00F80E	MOV.W #0280, SP	
00F80F	MOV.W #0280, SP	
00F810	MOV.W #0280, SP	
00F811	MOV.W #0280, SP	
00F812	MOV.W #0280, SP	
00F813	MOV.W #0280, SP	
00F814	MOV.W #0280, SP	
00F815	MOV.W #0280, SP	
00F816	MOV.W #0280, SP	
00F817	MOV.W #0280, SP	
00F818	MOV.W #0280, SP	
00F819	MOV.W #0280, SP	
00F81A	MOV.W #0280, SP	
00F81B	MOV.W #0280, SP	
00F81C	MOV.W #0280, SP	
00F81D	MOV.W #0280, SP	
00F81E	MOV.W #0280, SP	
00F81F	MOV.W #0280, SP	
00F820	MOV.W #0280, SP	
00F821	MOV.W #0280, SP	
00F822	MOV.W #0280, SP	
00F823	MOV.W #0280, SP	
00F824	MOV.W #0280, SP	
00F825	MOV.W #0280, SP	
00F826	MOV.W #0280, SP	
00F827	MOV.W #0280, SP	
00F828	MOV.W #0280, SP	
00F829	MOV.W #0280, SP	
00F82A	MOV.W #0280, SP	
00F82B	MOV.W #0280, SP	
00F82C	MOV.W #0280, SP	
00F82D	MOV.W #0280, SP	
00F82E	MOV.W #0280, SP	

