Practice Final Exam

一、True or False

1. (10 pts) Circle T (true) or F (false) for each of the following statements about the MSP430 processor.

T *F* (*a*) Register R2 is used for the stack pointer.

T *F* (*b*) Little endian refers to the storage order of data in the registers.

T *F* (*c*) An example of immediate addressing mode is mov.b #0xFF, R7.

T *F* (*d*) The MAR is in the ALU.

T *F* (*e*) The mov instruction has a default size of word.

T *F* (*f*) jn will jump if N=0

T *F* (*g*) The V flag is found in the Status Register (SR).

T *F* (*h*) The swpb instruction will affect condition bits.

T *F* (*i*) The program counter contains the location of the next instruction to be executed.

T *F* (*j*) Directive is part of the processor instruction set.

二、问答题

Write the definition and explain the purpose (what & why):

(a) ALU

(b) SR

(c) ISR

(d) condition codes

三、问答题

Trace through this assembly language program and show the final value of the affected registers and the memory at the end of execution. Find the memory and register values in **hexadecimal**.

```
        .data
count:  .byte  5
scores: .byte  33,22,48,11,34
info:   .space 1

        .text
        mov.w #count,R7
        mov.b @R7,R8
        clr R9
        clr R10
more: add.b scores(R9),R10

        add.w #1,R9
        cmp.w R8,R9  ;
        jnz   more
        mov.b R10,info
        jmp  $
```

四、问答题

This program uses R4 for a counter. Find the value of R4 at the end of the program for each of the 4 test conditions (jXX = jge, jz, jn, jnz), put your answers at right.

```
    .text
    clr.w  R11; counter
    mov.w  #3,R12 ; for loop?
again:
    inc.w  R11; increment the counter
    tst.w  R12
    jXX     done
    sub.w  #1, R12
    jmp    again

done:  jmp $     ; infinite loop to end
```

for jge, R11 = _____

for jz, R11 = _____

for jn, R11 = _____

for jnz, R11 =_____

五、问答题

Translate this high-level language into assembly language.

if T is greater than 10
then X=3
else Y=6
end if

The labels 'T', 'X', and 'Y' have already been defined as references to specific memory addresses, complete the rest of the assembly language program.

六、问答题

The MSP430 is connected to 2 LEDs and 2 switches. It has LED1 connected to P4.7, LED2 connected to P1.1, switch S1 connected to P1.6, and switch S2 connected to p1.7. Comment each line of the program.

```
#include <msp430.h>
int main(void)
{
   WDTCTL = WDTPW + WDTHOLD;       //_____________________________
   P4DIR |= BIT7;                  //_____________________________
   TA0CCTL0 = CCIE;                //_____________________________
   TA0CCR0 = 50000;
   TA0CTL = TASSEL_2 + MC_1 + TACLR;          //_____________________________
   __enable_interrupt();           //_____________________________
}
#pragma vector=TIMER0_A0_VECTOR    //_____________________________
__interrupt void TIMER0_A0_ISR(void)   //_____________________________
{
   P4OUT ^= BIT7;                  //_____________________________
}
```

Describe what this program does overall (which switches and LEDs are affected?)

七、问答题

The MSP430 is connected to 2 LEDs and 2 switches. It has LED1 connected to P4.7, LED2 connected to P1.1, switch S1 connected to P1.6, and switch S2 connected to p1.7. Comment each line of the program.

Write a program to blink LED1 at a rate of 1 Hz (once per second).

- When button S1 is pressed, LED1 stops blinking and turns off
- When button S1 is pressed again, LED1 starts blinking again

```
#include <msp430.h>
int main(void) {

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

    ____________________________________________________

}
```

```
#pragma vector =TIMER0_A0_VECTOR
__interrupt void TIMERA0_CCR0_ISR(void)
{
   _____________________________________________________

   _____________________________________________________

   _____________________________________________________

}

#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)

   {

   _____________________________________________________

   _____________________________________________________

   _____________________________________________________

   _____________________________________________________

   _____________________________________________________

   _____________________________________________________

   _____________________________________________________


}
```

Table 3–17. MSP430 Instruction Set

Mnemonic		Description		V	N	Z	C
ADC(.B)†	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src,dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src,dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src,dst	Clear bits in destination	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	Set bits in destination	src .or. dst → dst	–	–	–	–
BIT(.B)	src,dst	Test bits in destination	src .and. dst	0	*	*	*
BR†	dst	Branch to destination	dst → PC	–	–	–	–
CALL	dst	Call destination	PC+2 → stack, dst → PC	–	–	–	–
CLR(.B)†	dst	Clear destination	0 → dst	–	–	–	–
CLRC†		Clear C	0 → C	–	–	–	0
CLRN†		Clear N	0 → N	–	0	–	–
CLRZ†		Clear Z	0 → Z	–	–	0	–
CMP(.B)	src,dst	Compare source and destination	dst – src	*	*	*	*
DADC(.B)†	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src,dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)†	dst	Decrement destination	dst – 1 → dst	*	*	*	*
DECD(.B)†	dst	Double-decrement destination	dst – 2 → dst	*	*	*	*
DINT†		Disable interrupts	0 → GIE	–	–	–	–
EINT†		Enable interrupts	1 → GIE	–	–	–	–
INC(.B)†	dst	Increment destination	dst +1 → dst	*	*	*	*
INCD(.B)†	dst	Double-increment destination	dst+2 → dst	*	*	*	*
INV(.B)†	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		–	–	–	–
JEQ/JZ	label	Jump if equal/Jump if Z set		–	–	–	–
JGE	label	Jump if greater or equal		–	–	–	–
JL	label	Jump if less		–	–	–	–
JMP	label	Jump	PC + 2 x offset → PC	–	–	–	–
JN	label	Jump if N set		–	–	–	–
JNC/JLO	label	Jump if C not set/Jump if lower		–	–	–	–
JNE/JNZ	label	Jump if not equal/Jump if Z not set		–	–	–	–
MOV(.B)	src,dst	Move source to destination	src → dst	–	–	–	–
NOP†		No operation		–	–	–	–
POP(.B)†	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	–	–	–	–
PUSH(.B)	src	Push source onto stack	SP – 2 → SP, src → @SP	–	–	–	–
RET†		Return from subroutine	@SP → PC, SP + 2 → SP	–	–	–	–
RETI		Return from interrupt		*	*	*	*
RLA(.B)†	dst	Rotate left arithmetically		*	*	*	*
RLC(.B)†	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)†	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC†		Set C	1 → C	–	–	–	1
SETN†		Set N	1 → N	–	1	–	–
SETZ†		Set Z	1 → C	–	–	1	–
SUB(.B)	src,dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		–	–	–	–
SXT	dst	Extend sign		0	*	*	*
TST(.B)†	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

Port Registers

12.4.3 P1IES Register

Port 1 Interrupt Edge Select Register

Figure 12-3. P1IES Register

7	6	5	4	3	2	1	0
P1IES							
rw	rw	rw	rw	rw	rw	rw	rw

Table 12-5. P1IES Register Description

Bit	Field	Type	Reset	Description
7-0	P1IES	RW	undefined	Port 1 interrupt edge select 0b = P1IFG flag is set with a low-to-high transition. 1b = P1IFG flag is set with a high-to-low transition.

12.4.4 P1IE Register

Port 1 Interrupt Enable Register

Figure 12-4. P1IE Register

7	6	5	4	3	2	1	0
P1IE							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 12-6. P1IE Register Description

Bit	Field	Type	Reset	Description
7-0	P1IE	RW	0h	Port 1 interrupt enable 0b = Corresponding port interrupt disabled 1b = Corresponding port interrupt enabled

12.4.5 P1IFG Register

Port 1 Interrupt Flag Register

Figure 12-5. P1IFG Register

7	6	5	4	3	2	1	0
P1IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 12-7. P1IFG Register Description

Bit	Field	Type	Reset	Description
7-0	P1IFG	RW	0h	Port 1 interrupt flag 0b = No interrupt is pending 1b = Interrupt is pending

12.4.9 PxIN Register

Port x Input Register

Figure 12-9. PxIN Register

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

Table 12-11. PxIN Register Description

Bit	Field	Type	Reset	Description
7-0	PxIN	R	undefined	Port x input. Read only.

12.4.10 PxOUT Register

Port x Output Register

Figure 12-10. PxOUT Register

7	6	5	4	3	2	1	0
PxOUT							
rw	rw	rw	rw	rw	rw	rw	rw

Table 12-12. PxOUT Register Description

Bit	Field	Type	Reset	Description
7-0	PxOUT	RW	undefined	Port x output When I/O configured to output mode: 0b = Output is low 1b = Output is high When I/O configured to input mode and pullups/pulldowns enabled: 0b = Pulldown selected 1b = Pullup selected

12.4.11 PxDIR Register

Port x Direction Register

Figure 12-11. PxDIR Register

7	6	5	4	3	2	1	0
PxDIR							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 12-13. PxDIR Register Description

Bit	Field	Type	Reset	Description
7-0	PxDIR	RW	0h	Port x direction 0b = Port configured as input 1b = Port configured as output

Table 12-1. I/O Configuration

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

12.4.12 PxREN Register

Port x Pullup/Pulldown Resistor Enable Register

Figure 12-12. PxREN Register

7	6	5	4	3	2	1	0
PxREN							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 12-14. PxREN Register Description

Bit	Field	Type	Reset	Description
7-0	PxREN	RW	0h	Port x pullup or pulldown resistor enable. When respective port is configured as input, setting this bit will enable the pullup or pulldown. See Table 12-1 0b = Pullup or pulldown disabled 1b = Pullup or pulldown enabled

12.4.13 PxDS Register

Port x Drive Strength Register

Figure 12-13. PxDS Register

7	6	5	4	3	2	1	0
PxDS							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 12-15. PxDS Register Description

Bit	Field	Type	Reset	Description
7-0	PxDS	RW	0h	Port x drive strength 0b = Reduced output drive strength 1b = Full output drive strength

12.4.14 PxSEL Register

Port x Port Select Register

Figure 12-14. PxSEL Register

7	6	5	4	3	2	1	0
PxSEL							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 12-16. PxSEL Register Description

Bit	Field	Type	Reset	Description
7-0	PxSEL	RW	0h	Port x function selection 0b = I/O function is selected 1b = Peripheral module function is selected

Timer A Registers

Acronym	Register Name
TAxCTL	Timer_Ax Control
TAxCCTL0	Timer_Ax Capture/Compare Control 0
TAxCCTL1	Timer_Ax Capture/Compare Control 1
TAxCCTL2	Timer_Ax Capture/Compare Control 2
TAxCCTL3	Timer_Ax Capture/Compare Control 3
TAxCCTL4	Timer_Ax Capture/Compare Control 4
TAxCCTL5	Timer_Ax Capture/Compare Control 5
TAxCCTL6	Timer_Ax Capture/Compare Control 6
TAxR	Timer_Ax Counter
TAxCCR0	Timer_Ax Capture/Compare 0
TAxCCR1	Timer_Ax Capture/Compare 1
TAxCCR2	Timer_Ax Capture/Compare 2
TAxCCR3	Timer_Ax Capture/Compare 3
TAxCCR4	Timer_Ax Capture/Compare 4
TAxCCR5	Timer_Ax Capture/Compare 5
TAxCCR6	Timer_Ax Capture/Compare 6
TAxIV	Timer_Ax Interrupt Vector
TAxEX0	Timer_Ax Expansion 0

17.3.2 TAxR Register

Timer_Ax Counter Register

Figure 17-17. TAxR Register

15	14	13	12	11	10	9	8
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 17-5. TAxR Register Description

Bit	Field	Type	Reset	Description
15-0	TAxR	RW	0h	Timer_A register. The TAxR register is the count of Timer_A.

17.3.1 TAxCTL Register

Timer_Ax Control Register

Figure 17-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit clears TAR, the clock divider logic (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Table 17-1. Timer Modes

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

17.3.3 TAxCCTLn Register

Timer_Ax Capture/Compare Control n Register

Figure 17-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Table 17-6. TAxCCTLn Register Description

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections. 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read from this bit.
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

17.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

Figure 17-19. TAxCCRn Register

15	14	13	12	11	10	9	8
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 17-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed.

17.3.5 TAxIV Register

Timer_Ax Interrupt Vector Register

Figure 17-20. TAxIV Register

15	14	13	12	11	10	9	8
TAIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
TAIV							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

Table 17-8. TAxIV Register Description

Bit	Field	Type	Reset	Description
15-0	TAIV	R	0h	Timer_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest 04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG 06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG 08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG 0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG 0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG 0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest

17.3.6 TAxEX0 Register

Timer_Ax Expansion 0 Register

Figure 17-21. TAxEX0 Register

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved					TAIDEX(1)		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

(1) After programming TAIDEX bits and configuration of the timer, set TACLR bit to ensure proper reset of the timer divider logic.

Table 17-9. TAxEX0 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Reads as 0.
2-0	TAIDEX	RW	0h	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8