CPE 323
Intro to Embedded Computer Systems
MSP430 Instruction Set Architecture

Aleksandar Milenkovic

milenka@uah.edu

Admin

1. HW.1
2. HW.2
3. Quiz.02

MSP430 Instruction Set Architecture

- 1. Types of ISA (16, 16-bit GPRs, R0=PC, R1=SP, R2=SR, R3=CG)
- 2. Memory View (byte addressable, 16-bit word aligned, little-endian)
- 3. Data Types (8-bit, 16-bit numbers)
- 4. Instruction Set
- 5. Addressing Modes
- 6. Instruction Encoding
- 7. Exceptions

Review: Address Specifiers (As, Ad)

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/–	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/–	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

Instruction Formats

- Double-operand

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1st IW	opcode				S-reg				Ad	B/W	As		D-reg			
2nd IW (optional)	source addressing mode specific (offset/address)															
3rd IW (optional)	destination addressing mode specific (offset/address)															

- Single-operand

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1st IW	0	0	0	1	0	0	opcode			B/W	Ad		D-reg			
2nd IW (optional)	source addressing mode specific (offset/address)															

- Jumps

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1st IW	0	0	1	condition			10-bit offset									

Double Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst – src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (decimally)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	*
BIC(.B)	src,dst	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	*
AND(.B)	src,dst	src .and. dst → dst	0	*	*	*

Decimal ADD

* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set

Single Operand Instructions

Rotate-right through carry

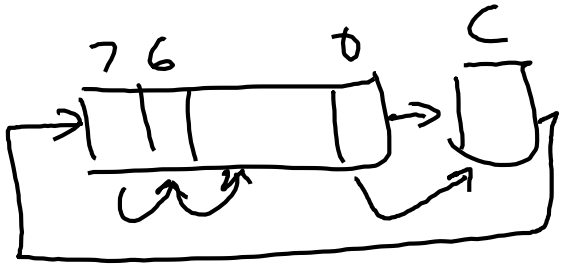
Mnemonic	S-Reg, D-Reg	Operation	Status Bits V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP – 2 → SP, src → @SP	–	–	–	–
SWPB	dst	Swap bytes	–	–	–	–
CALL	dst	SP – 2 → SP, PC+2 → @SP dst → PC	–	–	–	–
RETI		TOS → SR, SP + 2 → SP TOS → PC,SP + 2 → SP	*	*	*	*
SXT	dst	Bit 7 → Bit 8.....Bit 15	0	*	*	*

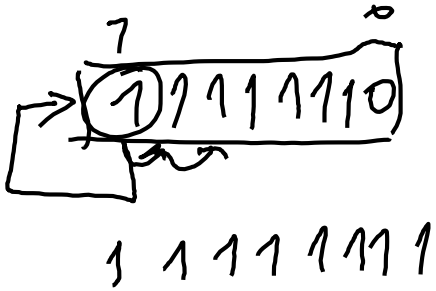
* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set





Jump Instructions

- NewPC <= PC + 2 + Offset*2

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1st IW	0	0	1	condition			10-bit offset									

[-512, 511]

-1024 1022

Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

MOV #myLab, PC

R0

add: dst <- dst + src

- add.b r5, r7

Add

r5	0x42CE
r7	0x200F

00 DD

0 1
CE neg.
0F pos
DD
1101 1101

C = 0
N = 1
Z = 0
V = 0

- add.w r5, r7

r5	0x42CE
r7	0x200F

62DD

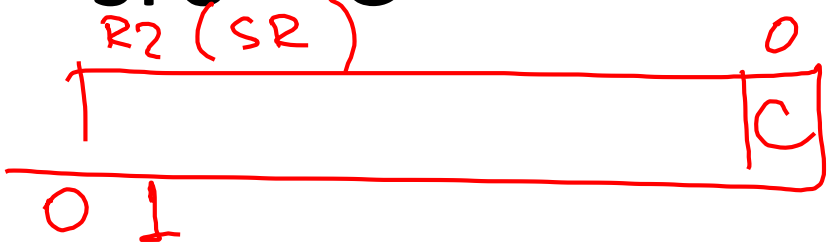
0001
0100
0010
42CE
200F
62DD
0110

C = 0
N = 0
Z = 0
V = 0

addc: : dst <- dst + src + C

- addc.b r5, r7

C = 1



r5	0x42CE
r7	0x200F

00 DE

```
  CE
+ 0F
+  1
----
  DE
```

C = 0
V = 0
N = 1
Z = 0

- addc.w r5, r7

r5	0x42CE
r7	0x200F

62DE

```
42CE
200F
   1
----
62DE
```

C = 0
V = 0
N = 0
Z = 0

dadd: dst <- dst + src + C (decimally)

- dadd.b r5, r7
- C=1

r5	0x4247
r7	x0203

0051

0 1 1
47
03
51

C=0
Z=0

- dadd.w r5, r7

r5	0x4247
r7	0x0203

4451

00011
4247
0203
4451

C=0
Z=0

sub: dst <- dst + #src + 1

- sub.b r5, r7 ; r7 ← r7 − r5

r5	0x42CE	
r7	x0245	0077

- sub.w r5, r7

r5	0x42CE	BD31
r7	0x0245	

r7 − r5
positive − positive

dst: 45
src: +31
+ 1
77

C = 0
Z = 0 V = 0
N = 0

0x CE
1100 1110
MAC: 00 11 0001

0x CE
31
+ 1
32
− 0x32
− 50

0 0 0 0
0245
+ BD31
+ 1
BF77
1011

C = 0
N = 1
Z = 0
V = 0

bit: src AND dst

- bit.b r5, r7

r5	0x42CE	
r7	x0245	0x0245

no changes

CE
45

1100 1110
0100 0101
01000100

C = 0
V = 0
Z = 0
N = 0

- bit.w r5, r7

?

r5	0x42CE	
r7	0x0245	0x0245

bic: dst ← #src AND dst

- bic.b r5, r7

r5	0x42CE	
r7	x0245	0x00 0 1

- bic.w r5, r7

r5	0x42CE	
r7	0x0245	

bis: dst ← src OR dst

- bis.b r5, r7

bitwise OR

dst: 45 | 01000101
src: CE | 11001110
1100,1111

r5	0x42CE
r7	x0245

0x00 CF

N = 1
Z = ∅

- bis.w r5, r7

r5	0x42CE
r7	0x0245

X

xor: dst ← src XOR dst

bitwise XOR

```
 01000101
 11001110
 ---------
 10001011
```

Z=0
N=1

- xor.b r5, r7

r5	0x42CE
r7	x0245

0x008B

- xor.w r5, r7

r5	0x42CE
r7	0x0245

and: dst ← src AND dst

- and.b r5, r7

r5	0x42CE
r7	x0245

↓

0x0044

45: 0100 0101
CE: 1100 1110
0100 0100

- and.w r5, r7

r5	0x42CE
r7	0x0245

rrc: rotate right through carry

- rrc.b r7

r7	0x45A8	c	1
r7	0x00D4	c	0

- rrc.w r7

r7	0x45A8	c	1
r7	0xA2D4	c	0

A8: 1010 1000 → 1 → C

1101 0100 → 0 → C

45A8: 0100 0101 1010 1000 C 1

1010 0010 1101 0100 1 0

rra: rotate right arithmetically

- rra.b r7

r7	**0x45A8**	**C**	**1**
r7	0x00D4	**C**	0

A8: 1010 1000 → C: 1

1101 0100 0

- rra.w r7

r7	**0x45A8**	**C**	**1**
r7		**C**	

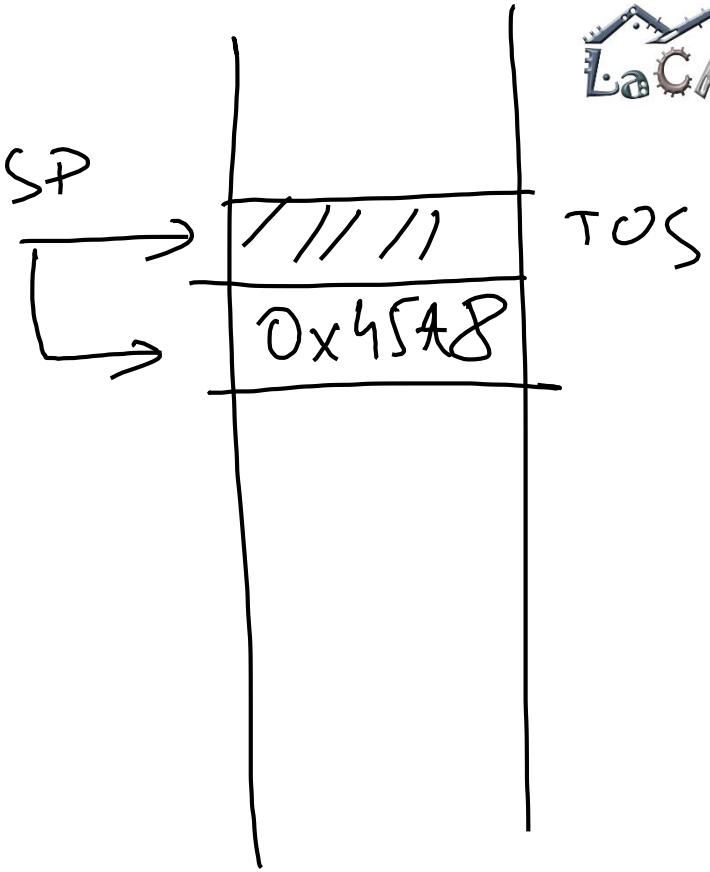
push, swpb

- push r7

; SP ← SP - 2

; M[SP] ← R7

r7 | 0x45A8



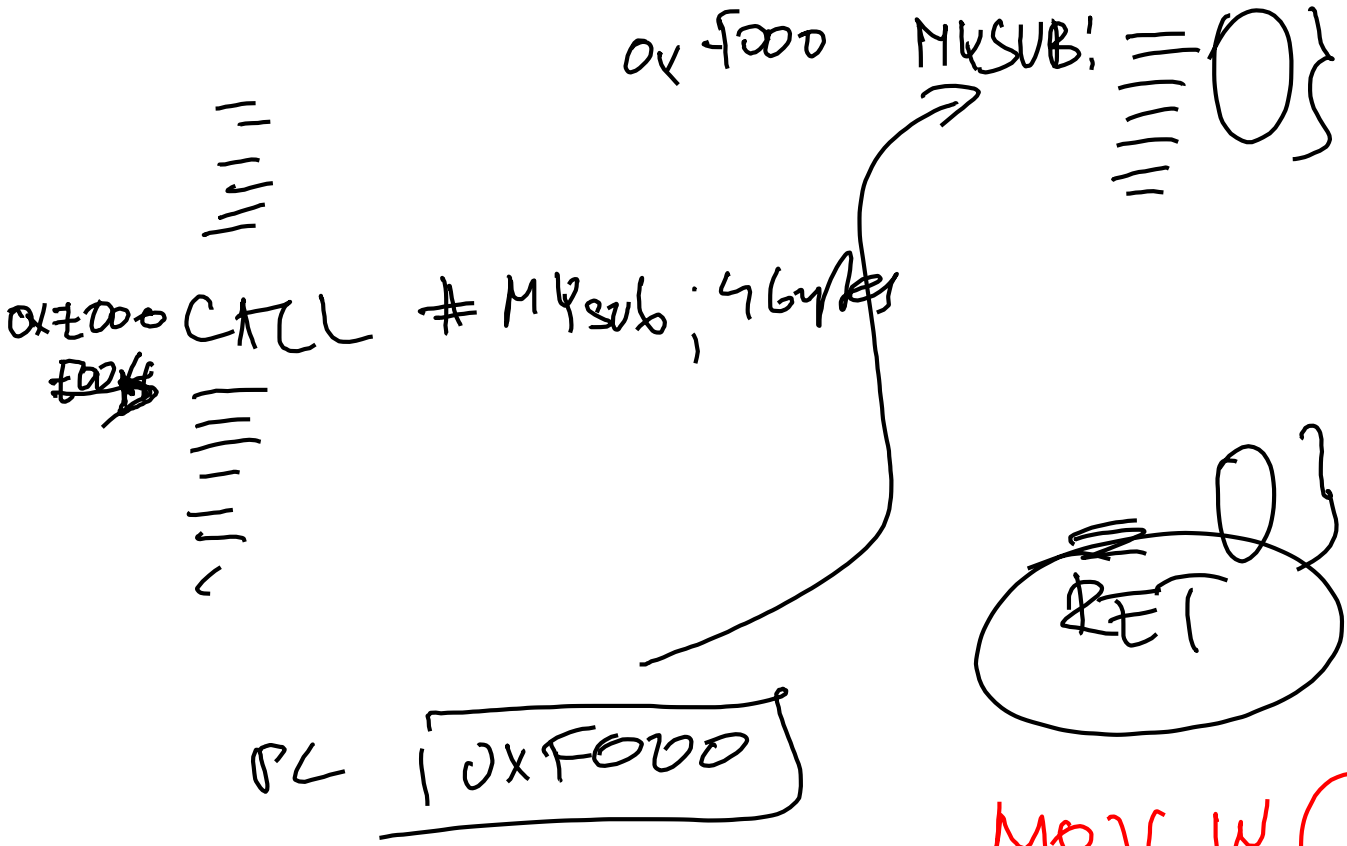
- swpb r7

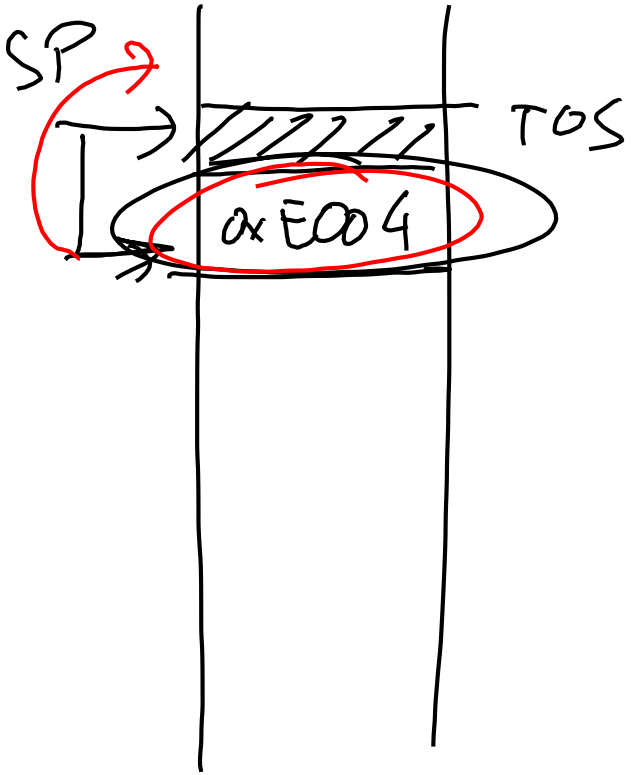
r7 | 0x45A8 | 0xA845

call

- call #mysub

SP ← SP - 2
M[SP] ← PC
PC ← #MYSUB





MOV.W @SP+, PC

reti

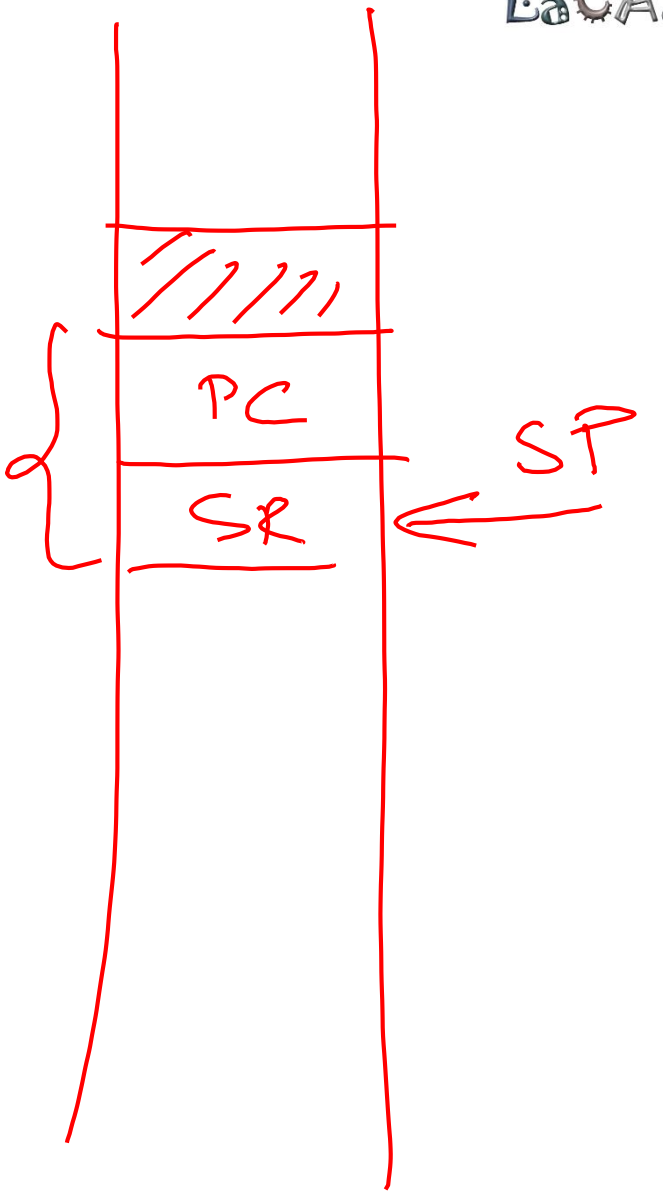
- reti

Exception

ISR

RETI → Return from Interrupt

SP ← M[SP]
SP ← SP + 2
PC ← M[SP]
SP ← SP + 2



Exceptions

- Asynchronous (triggered by in hardware)
- Synchronous (triggered in software, e.g., call OS function)
- Handled in Interrupt Service Routines or ISRs
  - Similar to subroutines, but no input or output parameters
- Exception processing
  - Occurs at the end of each instruction
  - Sequence of steps taken in hardware once interrupts (exceptions) are pending to determine which interrupts are pending and which one to service

HW structures for exceptions

- Interrupt Flag Registers: keep track of pending requests
- Interrupt Enable Register: allow for selective enabling/disabling of MASKABLE interrupts (control whether CPU sees them or not)
- Global Interrupt Enable (GIE) sits in SR (R2): disables all maskable interrupts
- Interrupt Vector Table: sits at the top of first 64 KB of address space and contains starting addresses of ISRs (defined by programmer/compiler)

Exception Processing (all steps are carried in hw)

- Finish current instruction
- Push PC and SR onto the stack
- Clear SR (exits low-power mode if CPU was in the one)
- If multiple interrupts are pending (more than one IFG bit is set), determine the highest priority one that is not masked to be served
  - Priority is determined by entries in IVT
- If single-source interrupt is serviced clear its IFG bit
- Read the starting address of the ISR from its entry in IVT and move it to PC (now you are in the ISR)