

CPE 323

Intro to Embedded Computer Systems

MSP430 Instruction Set Architecture

Aleksandar Milenkovic

milenska@uah.edu

Admin

1. HW.1

2. HW.2

3. Quiz.02

MSP430 Instruction Set Architecture

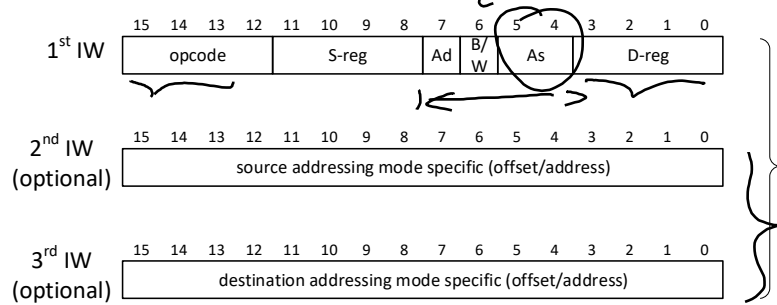
- 1. Types of ISA (16, 16-bit GPRs, R0=PC, R1=SP, R2=SR, R3=CG)
- 2. Memory View (byte addressable, 16-bit word aligned, little-endian)
- 3. Data Types (8-bit, 16-bit numbers)
- **4. Instruction Set**
- 5. Addressing Modes
- 6. Instruction Encoding
- **7. Exceptions**

Review: Address Specifiers (As, Ad)

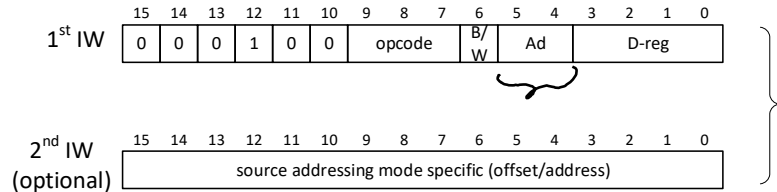
As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute 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

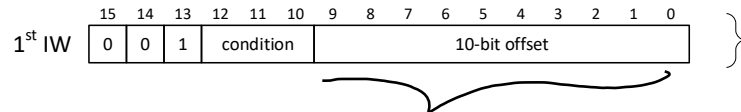
- Double-operand



- Single-operand



- Jumps



Double Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV (.B)	src, dst	<u>src</u> → 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	<u>.not.src</u> .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	*	*	*

* The status bit is affected

– The status bit is not affected

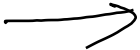
0 The status bit is cleared

1 The status bit is set

Decimal
ADD →

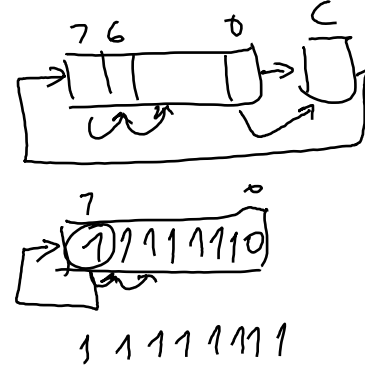
Single Operand Instructions

Rotate - right through
Carry



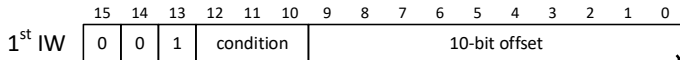
Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC (.B)	dst	C → MSB →LSB → C	*	*	*	*
RRA (.B)	dst	MSB → MSB →LSB → C	0	*	*	*
PUSH (.B)	src	SP - 2 → SP, src → @SP	-	-	-	-
SWPB	dst	Swap bytes	-	-	-	-
CALL	dst	SP - 2 → SP, PC+2 → @SP dst → PC	-	-	-	-
RETI		TOS → SR, SP + 2 → SP TOS → PC, SP +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

- $\text{NewPC} \leq \text{PC} + 2 + \text{Offset} * 2$



} $[-512, 511]$
 $-1024 \quad 1022$

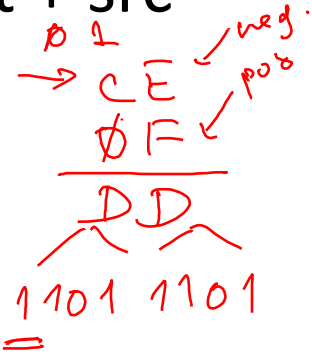
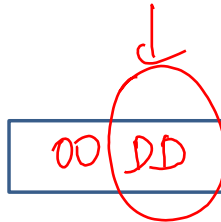
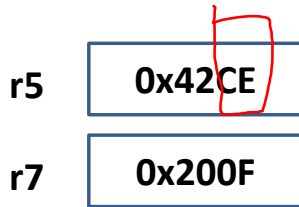
Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if $(N \cdot XOR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

mov #myLab, PC


add: dst <- dst + src

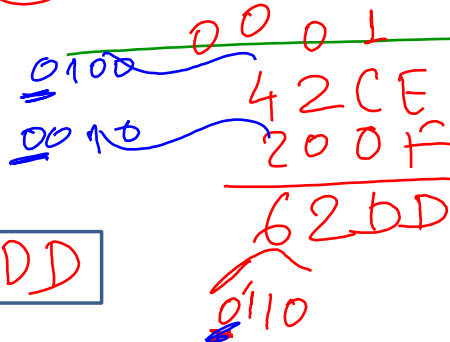
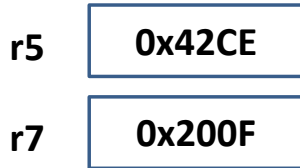
- add.b r5, r7

Add



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

- add.w r5, r7



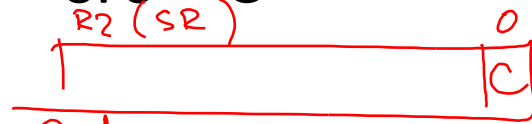
C = 0
N = 0
Z = 0
V = 0

addc: : dst <- dst + src + C

- addc.b r5, r7

C = 1

r5	0x42CE	
r7	0x200F	00 DE



01
CE
+ 0F
+ 1
DE

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

- addc.w r5, r7

r5	0x42CE	
r7	0x200F	62 DE

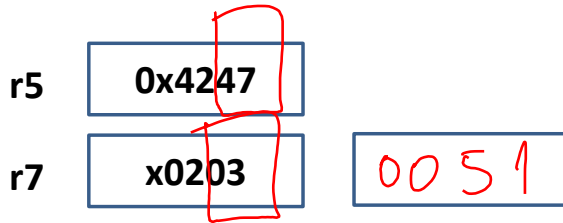
42 CE
200 F
1
62 DE

C = 0
V = 0
W = 0
Z = 0

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

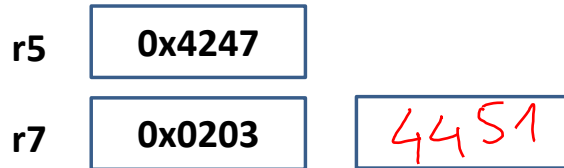
- dadd.b r5, r7

- C=1



$$\begin{array}{r}
 011 \\
 47 \\
 03 \\
 \hline
 51
 \end{array}
 \quad
 \begin{array}{l}
 C=0 \\
 Z=0
 \end{array}$$

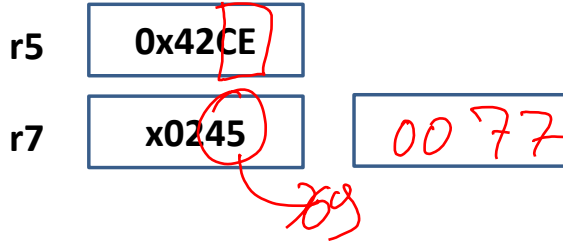
- dadd.w r5, r7



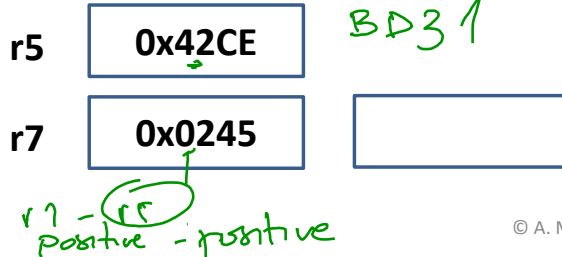
$$\begin{array}{r}
 0011 \\
 4247 \\
 0203 \\
 \hline
 4451
 \end{array}
 \quad
 \begin{array}{l}
 C=\phi \\
 Z=0
 \end{array}$$

sub: dst <- dst + #src + 1

- sub.b r5, r7 ; $r7 \leftarrow r7 - r5$



- sub.w r5, r7



Handwritten calculation for the sub.b instruction:

dst: 45

src: +31

+ 1

Handwritten status register values for the sub.b instruction:

C = 0

Z = 0

N = 0

V = 0

Handwritten calculation for the sub.w instruction:

0000

0245

+ BD31

+ 1

BF77

Handwritten calculation for the sub.w instruction:

0x CE

1100 1110

1100 1100 1

Handwritten calculation for the sub.w instruction:

0x CE

31

+ 1

32

- 0x32

- 50

Handwritten status register values for the sub.w instruction:

C = 0

N = 1

Z = 0

V = 0

bit: src AND dst

- bit.b r5, r7

r5 **0x42CE**
r7 **0x0245**

no changes
↓
0x0245

- bit.w r5, r7

r5 **0x42CE**
r7 **0x0245**

?
0x0245

CE
45

1100 1110
0100 0101

0100 0100

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

bic: dst ← #src AND dst

- bic.b r5, r7

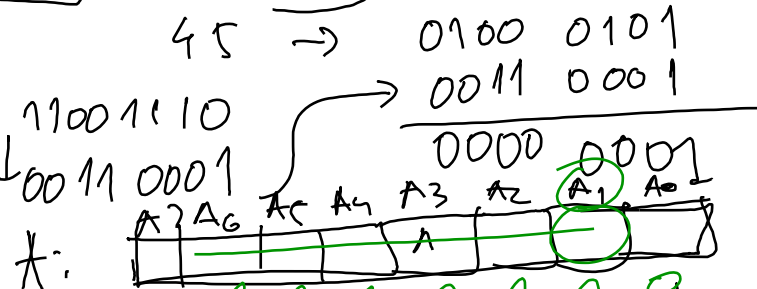
r5 **0x42CE**
r7 **x0245**

- bic.w r5, r7

r5 **0x42CE**
r7 **0x0245**

0x0001

CE → 11001110
1st comp: 00110001

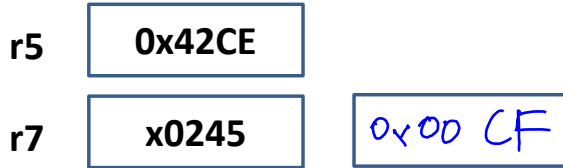


BICB #1, A
Immediate shift
0000-0001

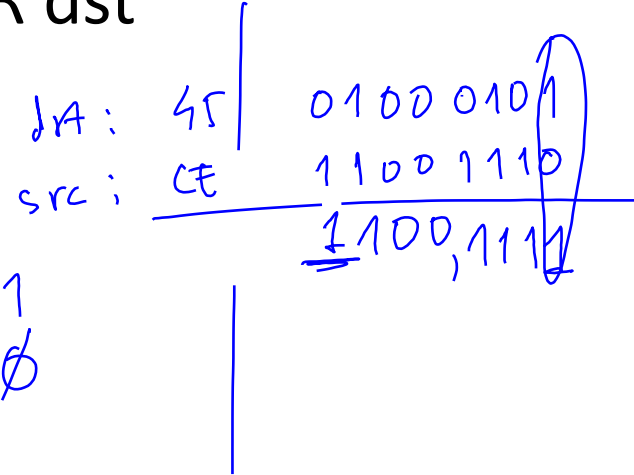
?

bis: dst ← src OR dst

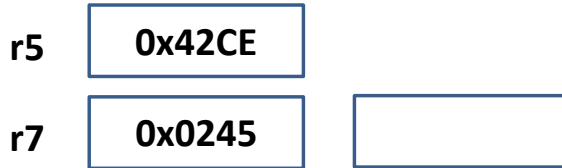
- bis.b r5, r7



bitwise
OR



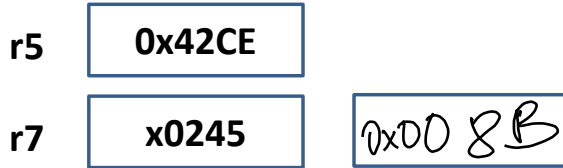
- bis.w r5, r7



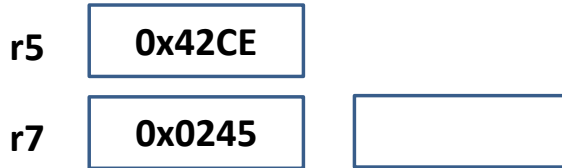
X

xor: dst \leftarrow src XOR dst

- xor.b r5, r7



- xor.w r5, r7



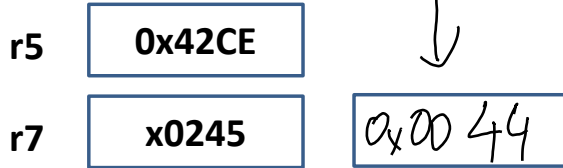
Bitwise XOR

0	1	0	0	0	1	0	1
1	1	0	0	1	1	1	0
<hr/>							
1	0	0	0	1	0	1	1

Z=0
N=1

and: $\text{dst} \leftarrow \text{src AND dst}$

- and.b r5, r7

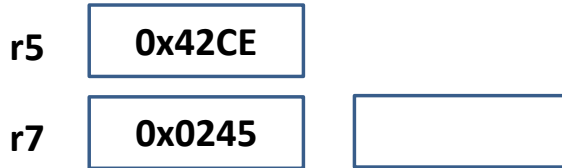


Handwritten binary representation of the AND operation:

```

r5:  0100 0101
CE:  1100 1110
-----
      0100 0100
  
```

- and.w r5, r7



rrc: rotate right through carry

- rrc.b(r7)

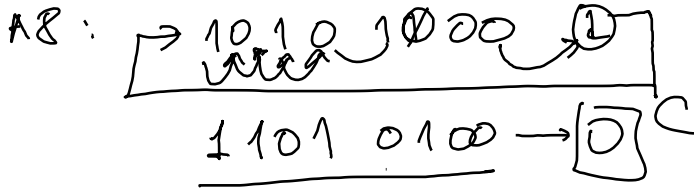
r7 0x45A8 c 1

r7 0x00D4 c 0

- rrc.w r7

r7 0x45A8 c 1

r7 0xA2D4 c 0



45A8:

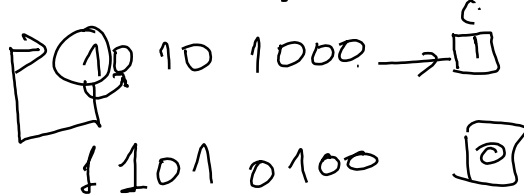
$$\begin{array}{r} 0100010110101000 \text{ (45A8)} \\ \xrightarrow{\text{rotate right}} 1010001011010100 \text{ (45A8)} \end{array}$$

rra: rotate right arithmetically

- rra.b r7

r7 0x45A8 c 1

r7 0x00D4 c 0

A8: 

- rra.w r7

r7 0x45A8 c 1

r7 c

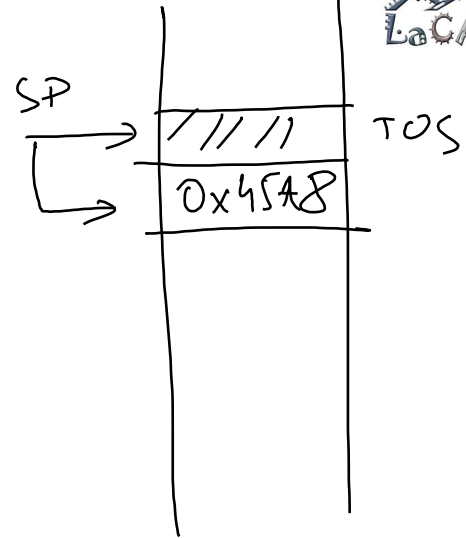
push, swpb

- push r7

r7

0x45A8

```
; SP ← SP - 2
; M[SP] ← R7
```



- swpb r7

r7

0x45A8

0xA845

call

- call #mysub

$SP \leftarrow SP - 2$
 $M[SP] \leftarrow PC$
 $PC \leftarrow \#MYSUB$



reti

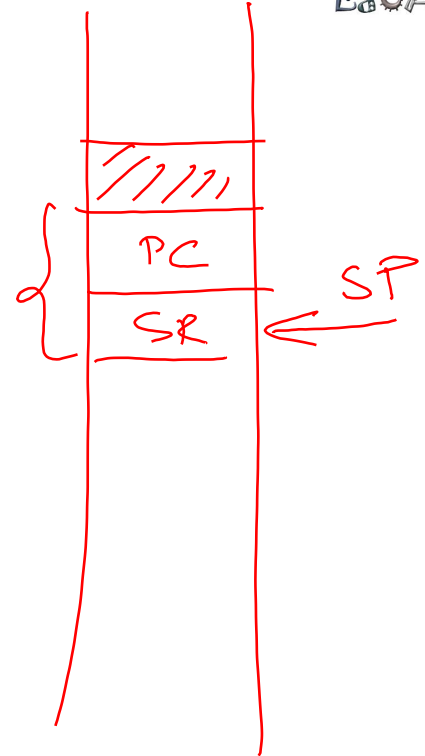
- reti

Exception

$ISP \leftarrow$

$SP \leftarrow M[SP]$
 $SP \leftarrow SP + 2$
 $PC \leftarrow M[SP]$
 $SP \leftarrow SP + 2$

Return
from
Interrupt



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)