

**Table 1a: The complete MSP430 instruction set of 27 core instructions**

core instruction mnemonics	core instruction binary															
Single-operand arithmetic	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
	5	4	3	2	1	0	opcode			B/W	As	source				
RRC Rotate right through carry	0	0	0	1	0	0	0	0	0	B/W	As	source				
SWPB Swap bytes	0	0	0	1	0	0	0	0	1	0	As	source				
RRA Rotate right arithmetic	0	0	0	1	0	0	0	1	0	B/W	As	source				
SXT Sign extend byte to word	0	0	0	1	0	0	0	1	1	0	As	source				
PUSH Push value onto stack	0	0	0	1	0	0	1	0	0	B/W	As	source				
CALL Subroutine call; push PC and move source to PC	0	0	0	1	0	0	1	0	1	0	As	source				
RETI Return from interrupt; pop SR then pop PC	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0
Conditional jump; PC = PC + 2×offset	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
	5	4	3	2	1	0	condition			10-bit signed offset						
JNE/JNZ Jump if not equal/zero	0	0	1	0	0	0	10-bit signed offset									
JEQ/JZ Jump if equal/zero	0	0	1	0	0	1	10-bit signed offset									
JNC/JLO Jump if no carry/lower	0	0	1	0	1	0	10-bit signed offset									
JC/JHS Jump if carry/higher or same	0	0	1	0	1	1	10-bit signed offset									
JN Jump if negative	0	0	1	1	0	0	10-bit signed offset									
JGE Jump if greater or equal (N == V)	0	0	1	1	0	1	10-bit signed offset									
JL Jump if less (N != V)	0	0	1	1	1	0	10-bit signed offset									
JMP Jump (unconditionally)	0	0	1	1	1	1	10-bit signed offset									
Two-operand arithmetic	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
	5	4	3	2	1	0	source		A/d	B/W	As	destination				
MOV Move source to destination	0	1	0	0	source		source		A/d	B/W	As	destination				
ADD Add source to destination	0	1	0	1	source		source		A/d	B/W	As	destination				
ADDC Add w/carry: dst += (src+C)	0	1	1	0	source		source		A/d	B/W	As	destination				
SUBC Subtract w/ carry: dst -= (src+C)	0	1	1	1	source		source		A/d	B/W	As	destination				
SUB Subtract; dst -= src	1	0	0	0	source		source		A/d	B/W	As	destination				
CMP Compare; (dst-src); discard result	1	0	0	1	source		source		A/d	B/W	As	destination				
DADD Decimal (BCD) addition: dst += src	1	0	1	0	source		source		A/d	B/W	As	destination				
BIT Test bits; (dst & src); discard result	1	0	1	1	source		source		A/d	B/W	As	destination				
BIC Bit clear; dest &= ~src	1	1	0	0	source		source		A/d	B/W	As	destination				
BIS "Bit set" - logical OR; dst  = src	1	1	0	1	source		source		A/d	B/W	As	destination				
XOR Bitwise XOR; dst ^= src	1	1	1	0	source		source		A/d	B/W	As	destination				
AND Bitwise AND; dst &= src	1	1	1	1	source		source		A/d	B/W	As	destination				

The source and destination of an instruction are defined by the following fields:

src The source operand defined by As and S-reg  
dst The destination operand defined by Ad and D-reg  
As The addressing bits responsible for the addressing mode used for the source (src)  
S-reg The working register used for the source (src)  
Ad The addressing bits responsible for the addressing mode used for the destination (dst)  
D-reg The working register used for the destination (dst)  
B/W Byte or word operation:  
0: word operation  
1: byte operation

Table 1b: The emulated instructions

emulated	core instructions	instructions	
ADC.x dst	ADDC.x #0,dst	add carry to destination	
CLRC	BIC #1,SR	clear carry bit	0xc312
CLRN	BIC #4,SR	clear negative bit	0xc222
CLRZ	BIC #2,SR	clear zero bit	0xc322
DADC.x dst	DADD.x #0,dst	decimal add carry to destination	
DEC.x dst	SUB.x #1,dst	decrement	
DECD.x dst	SUB.x #2,dst	double decrement	
DINT	BIC #8,SR	disable interrupts	0xc232
EINT	BIS #8,SR	enable interrupts	0xd232
INC.x dst	ADD.x #1,dst	increment	
INCD.x dst	ADD.x #2,dst	double increment	
INV.x dst	XOR.x #-1,dst	invert	
NOP	MOV #0,R3	no operation	0x4303
POP dst	MOV @SP+,dst	pop from stack	
RET	MOV @SP+,PC	return from subroutine	0x4130
RLA.x dst	ADD.x dst,dst	rotate left arithmetic (shift left 1 bit)	
RLC.x dst	ADDC.x dst,dst	rotate left through carry	
SBC.x dst	SUBC.x #0,dst	subtract borrow (1-carry) from destination	
SETC	BIS #1,SR	set carry bit	0xd312
SETN	BIS #4,SR	set negative bit	0xd222
SETZ	BIS #2,SR	set zero bit	0xd322
TST.x dst	CMP.x #0,dst	test destination	

**Table 2. Summary of addressing modes**

A s	src	Syntax	Description
0	n	Rn	Register direct. The operand is the contents of Rn.
1	n	x(Rn)	Indexed. The operand is in memory at address Rn+x.
2	n	@Rn	Register indirect. The operand is in memory at the address held in Rn.
3	n	@Rn+	Indirect autoincrement. As above, then the register is incremented by 1 or 2.
Addressing modes using R0 (PC)			
1	0(PC)	label	Symbolic. x(PC) The operand is in memory at address PC+x.
3	0(PC)	#x	Immediate. @PC+ The operand is the next word in the instruction stream.
Addressing modes using R2 (SP) as CG1 and R3 as CG2, constant generation			
0	2(SP)	-	register mode
1	2 (SP)	&LABEL	Absolute. The operand is in memory at address x.
2	2 (SP)	#4	Constant. The operand is the constant 4.
3	2 (SP)	#8	Constant. The operand is the constant 8.
0	3(R3)	#0	Constant. The operand is the constant 0.
1	3 (R3)	#1	Constant. The operand is the constant 1. There is no index word.
2	3 (R3)	#2	Constant. The operand is the constant 2.
3	3 (R3)	#-1	Constant. The operand is the constant -1.

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.

Table 3: Example program layout in memory

memory location	byte	
F810		
	00	
	22	
	00	
	F7	
	40	
F80A	F2	40F2 00F7 0022
	01	
	20	
	5A	
	80	
	40	
F804	B2	40B2 5A80 0120
	02	
	80	
	40	
F800	31	4031 0280 mov #0x280, sp

The table to the left shows how the binary executable is to be stored in the flash memory. Notice that it is usual to show memory locations with the lowest addresses at the bottom of the table. The memory locations available for the program start at f800. The information is organized in 2 bytes words. The low byte is stored in a location of the lower address and high byte of the word goes into the location with the higher address in the contiguous flash memory. This is called a little-endian format as shown in the table. (The big-endian format stores the high bytes in the lower addresses). Program code segment grows upward. The stack pointer is set at top of the ram, the bytes pushed into the stack go downward. Vectors are stored in a specially reserved flash memory locations.

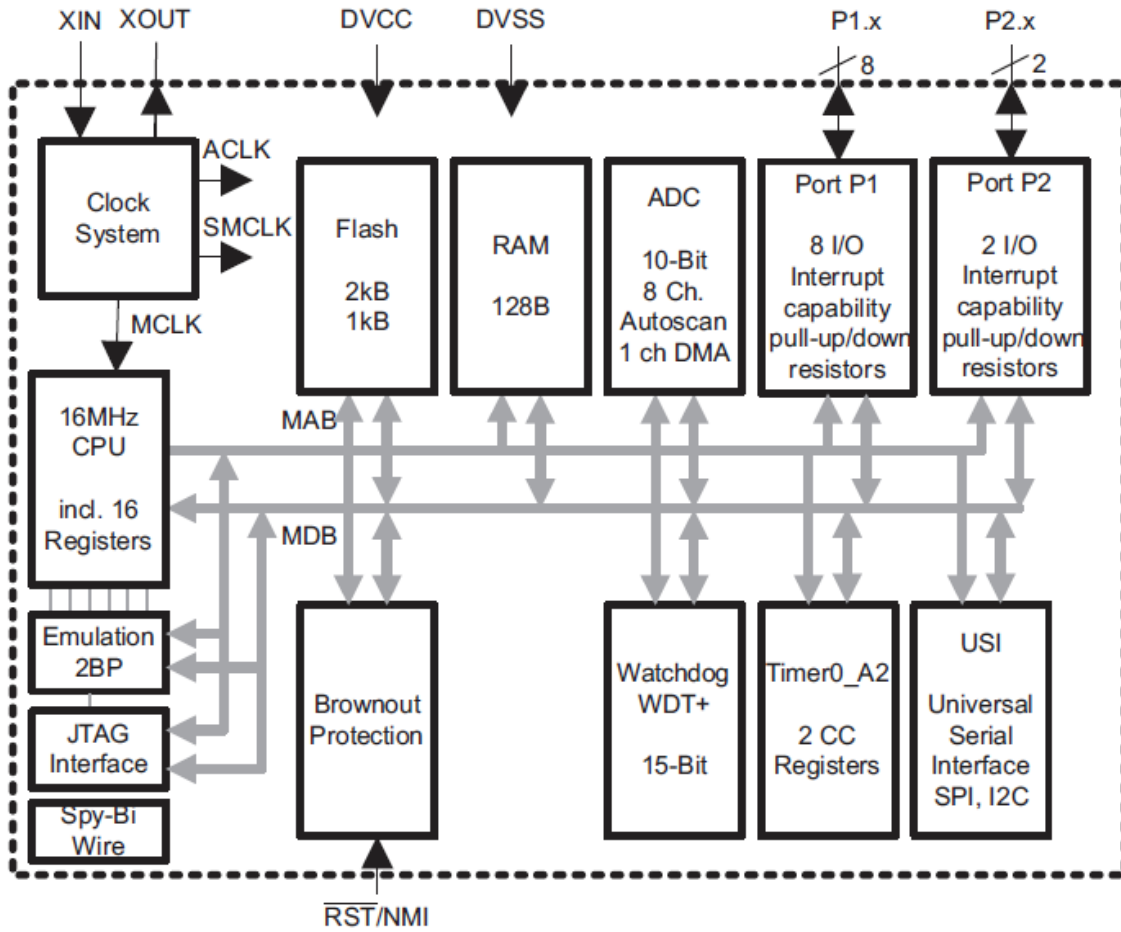
Table 4: Memory map of G2231

address in hex	memory map	functional blocks used	physical addresses
fff . . ffe0	interrupt vector table	power-up/main	ffe=f800
ffd . . f800	flash	program	org f800
10ff . . 1000	information memory (factory)		
027f . . 0200	ram	stack pointer	0280 (push = pre-decrement; pop=post-increment)
01ff . . 0100	16-bit peripherals	watchdog timer,	0120=5a80
00ff . . 0010	8-bit peripherals	port 1	P1OUT 0021=x P1DIR 0022=1
000f . . 0000	Special function registers		

**Table 5: Decimal/Binary/Hexadecimal Reference Table**

decimal	binary, b	0x, hexadecimal, h	<p>There are 3 popular ways of representing the numbers used in programming: our familiar decimal system, binary system (digits 0 and 1) and hexadecimal system (digits 0-9, a, b, c, d, e, f). Prefix 0x or suffix h is used to indicate hexadecimal numbers, suffix b is used to indicate binary. To convert a number from binary to hexadecimal (see the table) you should group the binary digits them into 4-bit groups and assign a hexadecimal digit to each group.</p> <p>When you prepare bit patterns to write into registers the binary notation, for example 11110111b is much more descriptive than the hexadecimal one 0xF3.</p> <p>Executable program is always stored in the computer memory as binary, to assemble the instruction is to convert them from the mnemonics format into binary, to disassemble is the reverse process. You will try both during lab 1..</p>
0	0000 0000	00	
1	0000 0001	01	
2	0000 0010	02	
3	0000 0011	03	
4	0000 0100	04	
5	0000 0101	05	
6	0000 0110	06	
7	0000 0111	07	
8	0000 1000	08	
9	0000 1001	09	
10	0000 1010	0A	
11	0000 1011	0B	
12	0000 1100	0C	
13	0000 1101	0D	
14	0000 1110	0E	
15	0000 1111	0F	
16	0001 0000	10	

### Functional Block Diagram, MSP430G2x31



Examples of Dissassembling a Command (using first 3 directives of lab1 program):

Instruction 1:

It is standard procedure is to initialize the RAM for stack operation using the instruction :

```
mov #0x280, sp
```

Use Table 1. To understand the binary format of this instruction shown below. This instruction moves the hexadecimal number 280 (0x280) to register sp which is a second register in the cpu memory map (Fig.2.),

- two-operand move opcode becomes 4 (0100b)
- source register (S-reg): we are using here the immediate addressing which means that the number is stored directly in the command. It is pointed out by the number in the program counter (PC) so that field becomes 0, (0000b). For immediate addressing mode the source register is always PC.
- for word bit B/W becomes 0 (this bit is 1 when the instruction deals with bytes of 8bits each or is 0 if the instruction deals with words of 16bits each)
- immediate mode As/Ad becomes 11/- (- means either 0 or 1), consult the table 2 of addressing modes
- so the field Ad B/W As is 0011b or 0x3 (hexadecimal 3)
- destination register (D-Reg) field of sp (stack pointer) is 1
- 0280 will be the operand which follows the instruction

MOV Move source to destination	0	1	0	0	S-reg				A	B/	As	D-Reg				
Mov.b #0x280, sp	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1
which disassemble to binary																

4031 0280 this hexadecimal number is the instruction in a format ready to send to the flush memory of the microprocessor.

Instruction 2

The following instruction will stop the watchdog timer:

```
mov.w # WDTPW|WDTHOLD, & WDTCTL
```

WDTPW and WDTHOLD are constants defined in the include file, the symbol | indicates that they are added to become 0x5A80, When send to the register at the address WDTCTL which is hexadecimal 0120 it stops the watchdog timer.

- two-operand move opcode, green columns, becomes 4`
- Source in immediate addressing is 0x0
- bit B/W becomes 0
- immediate mode As/Ad becomes 11/-
- destination is an absolute address specified in the command, D-Reg field becomes 2.
- 5A80 is the word to send to watchdog timer to stop it and 0120 is the watchdog timer's absolute address which is the operands which follows the instruction -

MOV Move source to destination	0	1	0	0	S-reg				A	B/	As	D-Reg				



mov.w #WDTPW WDTHOLD, &WDTCTL	0	1	0	0	0	0	0	0	1	0	1	1	0	0	1	0
-------------------------------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

40B2 5A80 0120 is a 6 byte instruction. It will take 5 cycles of CPU clock to execute.

Instruction 3:

To assemble the instruction

mov.b #11110111b, & P1DIR

- two-operand move opcode is 4
- byte B/W becomes 1 (byte operation)
- immediate mode As/Ad becomes 11/-
- source S-reg is implied to PC that field becomes 0
- destination is absolute address , D-Reg becomes 2
- #N 00f7 (binary 11110111) absolute address 0022 of port P1 direction register will be the operands which follows the instruction –

MOV Move source to destination	0	1	0	0	S-reg	A d	B/ W	As	D-Reg							
mov.b #11110111b, &P1DIR which disassemble to binary	0	1	0	0	0	0	0	0	1	1	1	1	0	0	1	0

40F2 00F7 0022

To assemble the instruction mov.b #0, & P1OUT

- We can use exactly the same format as the previous one and obtain: 40F2 0000 0021 where 0021 is the address of the P1OUT register
- but instead we can use the fact that certain numbers like 0 can be generated by a constant generation register As becomes 0, S-reg becomes 3, which results in faster more efficient code.

MOV Move source to destination	0	1	0	0	S-reg	A d	B/ W	As	D-Reg							
mov.b #0, &P1OUT which disassemble to binary	0	1	0	0	0	0	1	1	1	1	0	0	0	0	2	0

43C2 0021 is a 4 byte instruction. It will take 4 cycles of CPU clock to execute.

Similarly we can use constant #1 in assembling the instruction bis #1, &P1OUT will yield D3E2 0021. The list of As values for constant generation using source register 3 is in *Table 3–2. Values of Constant Generators CG1, CG2 of the MSP430x2xx Family User's Guide*

Use the instructions described above with appropriate values of the parameters to create a program, which when run by the microprocessor will display first 4 digits of your student number on the seven segment display.