Introduction TI MSP430 : ASM coding

Seb

Hackerspace Brussels

14 Jul 2012

What's a Processor

Definition from Wikipedia

It is a multipurpose, programmable device that accepts digital data as input, processes it according to instructions stored in its memory, and provides results as output. It is an example of sequential digital logic, as it has internal memory.
Microprocessors operate on numbers and symbols represented in the binary numeral system.

Why MSP430 ?

Advantages

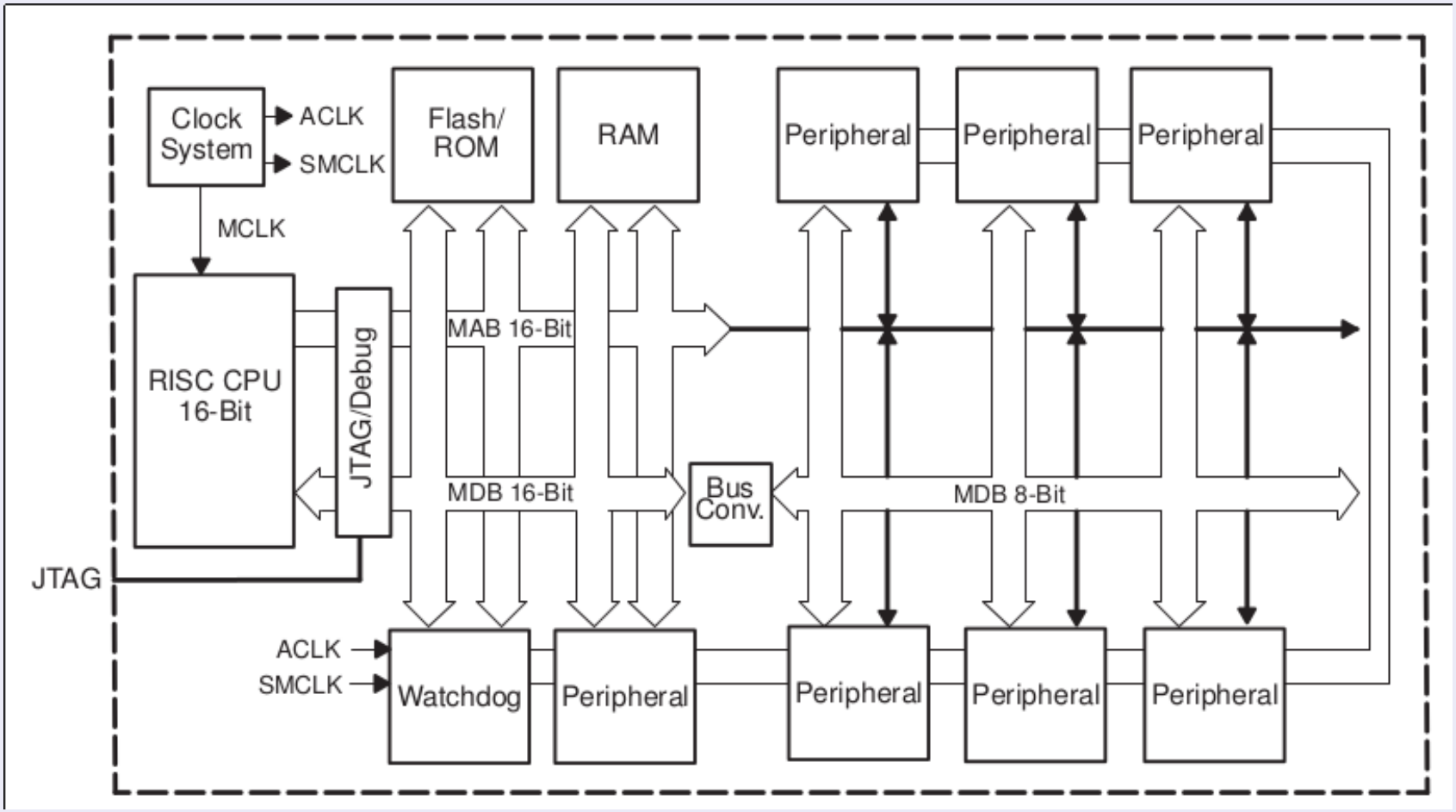
- 16 bits Microcontroller
- Low Cost (begins at 0.5$ and free samples ;)
- Launchpad (evaluation platform at 4.30$)
- Supported by GCC and Binutils
- Ultra low power consumption

Drawbacks

- Weird peripheral configuration
- Memory space limited to 64kB (MSP430) and 1MB (MSP430X)
- Low IPC count
- Little-Endian

How MSP430 and Processors work

Internals



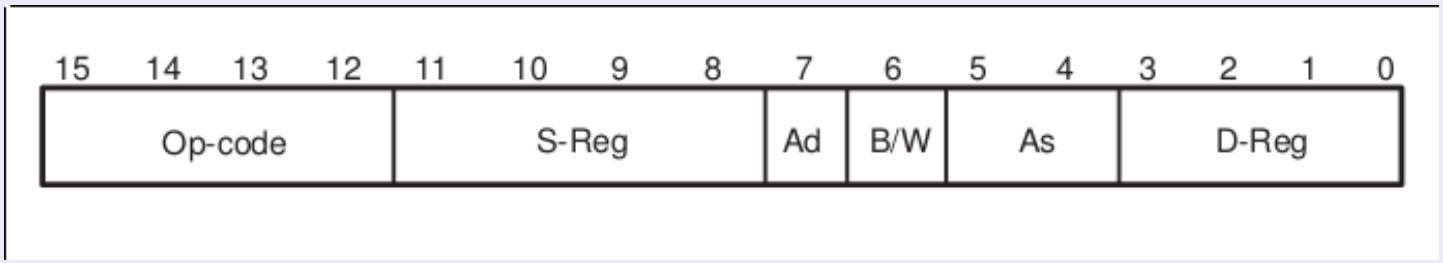
Instruction Cycle

Instruction Cycle

- Instruction Fetch (Get what you need to do)
- Instruction Decode (Understand what you need to do)
- First Operand Fetch (Not enough information, get some more)
- Second Operand Fetch (Still not enough information, get some more)
- Execute (Do it !)
- Writeback (Write result of the operation)

Instruction Format for 2 Operands Arithmetic

Instruction Encoding

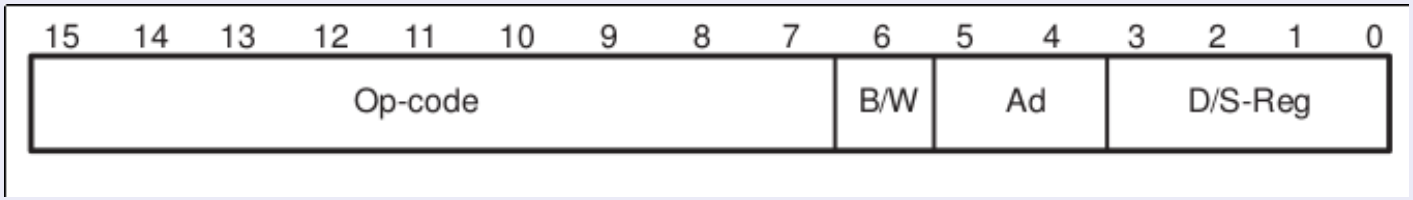


Instruction List

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	*	*	*

Instruction Format for 1 Operand Arithmetic

Instruction Encoding

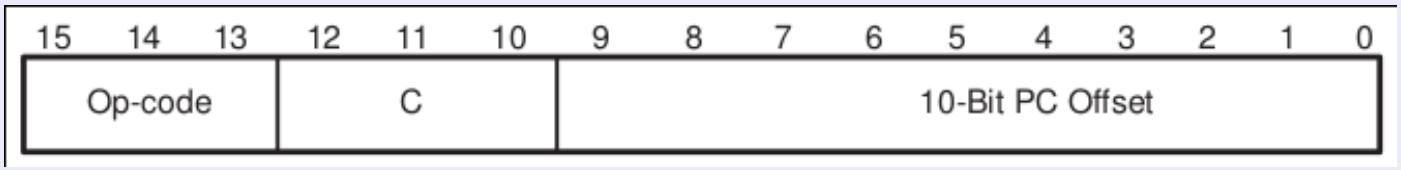


Instruction List

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	*	*	*

Instruction Format for Conditional Jump

Instruction Encoding

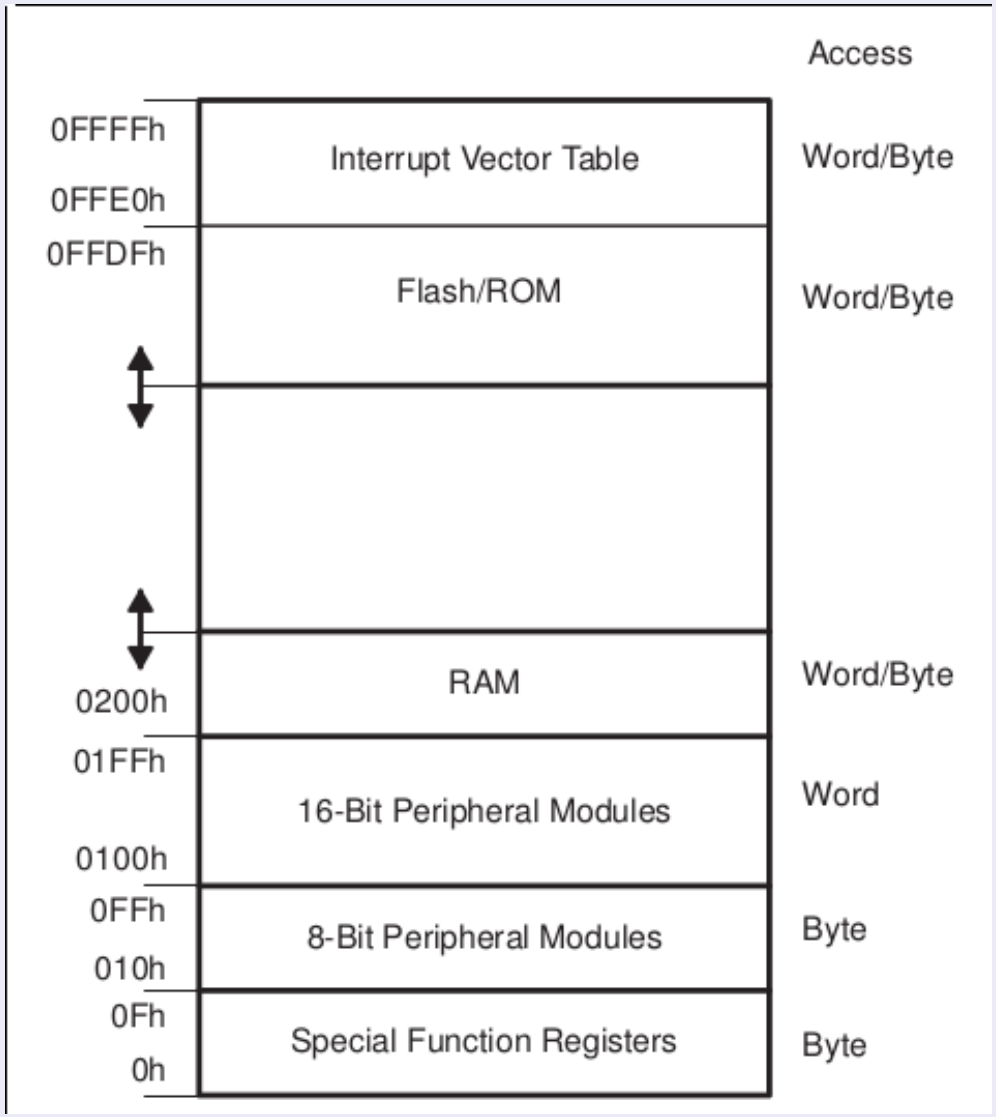


Instruction List

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

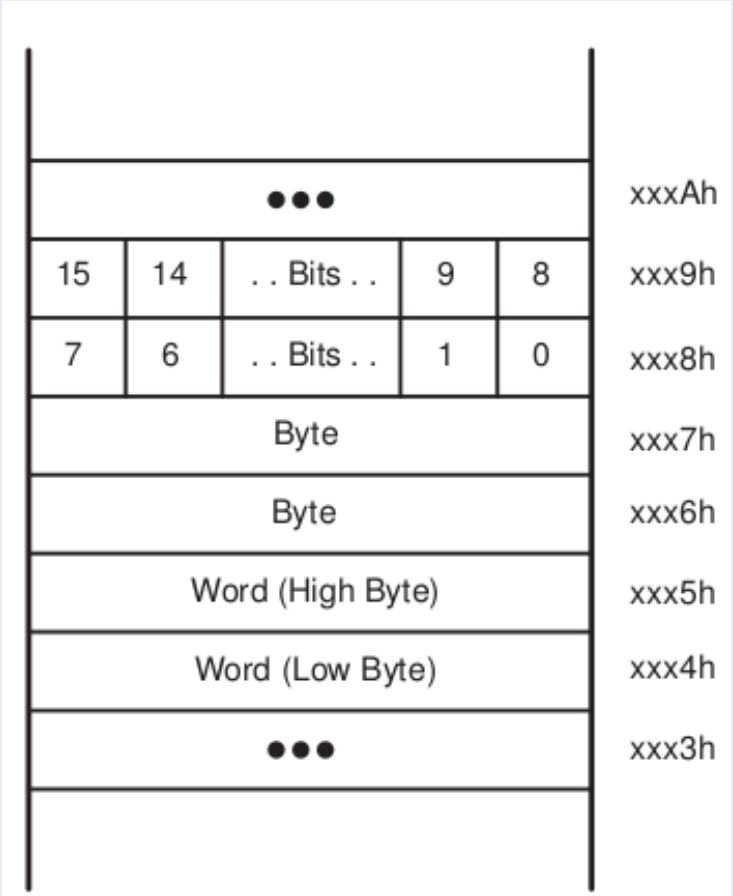
Memory Mapping

Memory Map



Memory Mapping

Byte Ordering (Little Endian)



Addressing Modes

Source Addressing Modes

- Register
- Indexed
- Symbolic (PC Relative)
- Absolute Address
- Indirect Register
- Indirect Autoincrement
- Immediate

Destination Addressing Modes

- Register
- Symbolic (PC Relative)
- Absolute Address
- Indexed

Addressing Modes : Register Mode

Example

mov R5, R6

Explanation

- Moves the content or the register R5 into R6 without altering R5.

Usefulness

- Save a register to another

Addressing Modes : Indexed Mode

Example

mov 4(R5), R6

Explaination

- Add 4 to the content of R5 inside the CPU
- Fetch the memory address from the forementionned computation
- Store the value into R6

Usefulness

- Access an item in memory (eg. an array) with a constant offset

Addressing Modes : Symbolic Mode

Example

mov 0x1234, R6

Explaination

- Add 0x1234 to the PC to generate the address
- Fetch the memory from the address of the forementionned computation
- Store the value into R6

Usefulness

- Access an array of data stored in the program memory

Addressing Modes : Absolute Mode

Example

mov &0xDEAD, R6

Explaination

- Fetch the memory from the address 0xDEAD
- Store the value into R6

Usefulness

- Access memory at a known address (eg. Peripheral)

Addressing Modes : Indirect Register Mode

Example

mov @R8, R6

Explaination

- Fetch the memory at the address contained in R8
- Store the value into R6

Usefulness

- Use a register as a pointer to memory

Addressing Modes : Indirect Autoincrement Mode

Example

mov @R8+, R6

Explaination

- Fetch the memory at the address contained in R8
- Store the value into R6
- Increment R8

Usefulness

- Copy a data to somewhere else in 1 instruction
- Stack Popping

Addressing Modes : Immediate Mode

Example

mov #0xBEEF, R6

Explanation

- Load R6 with 0xBEEF

Usefulness

- Initialize a register with a value

Word or Byte ?

Word Access

- CPU naturally works with 16 bits words.
- Instructions suffixed with .W or nothing.
- Flags are updated with the word operation.
- Address of the operands MUST be aligned on 16 bits (also true for C).

Byte Access

- Sometimes, it's necessary to work with bytes.
- Instructions are suffixed with .B.
- Flags are updates with the byte operation.

Registers

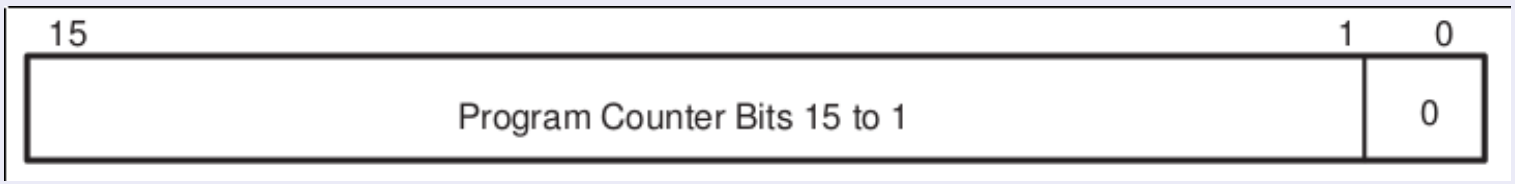
Registers Description

R0: Program Counter (Address of the next instruction)
R1: Stack Pointer
R2: Status Register (Generates other constants)
R3: Constant Generator (Generates 0, 1, 2, -1)
R4-R15: General Purpose Registers

Registers

R0 Program Counter

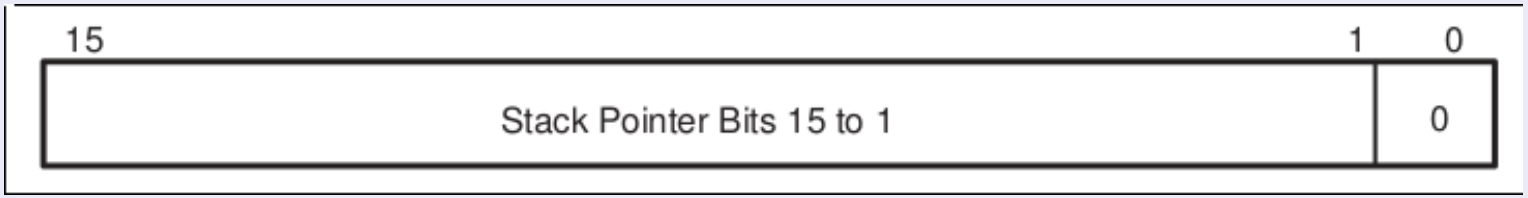
Contains the next instruction to be executed.



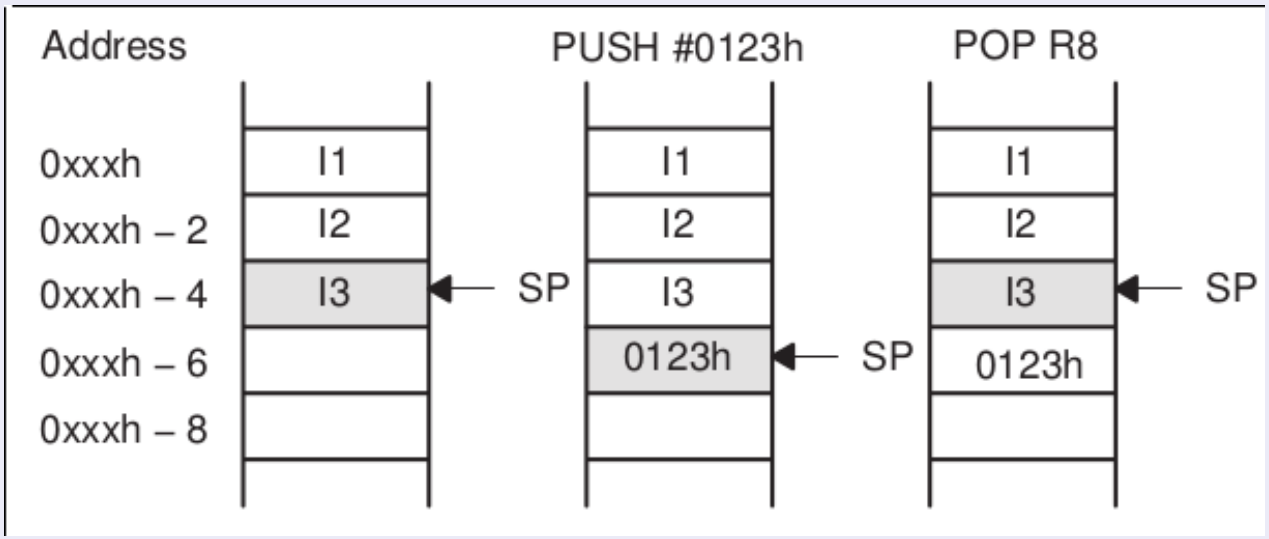
Registers

R1 Stack Pointer

Contains the next value where value will be stored.



Stack



Stack (LIFO) Management

Pushing

- Store temporary data
- Keep track of callings
- Save return status for interrupts

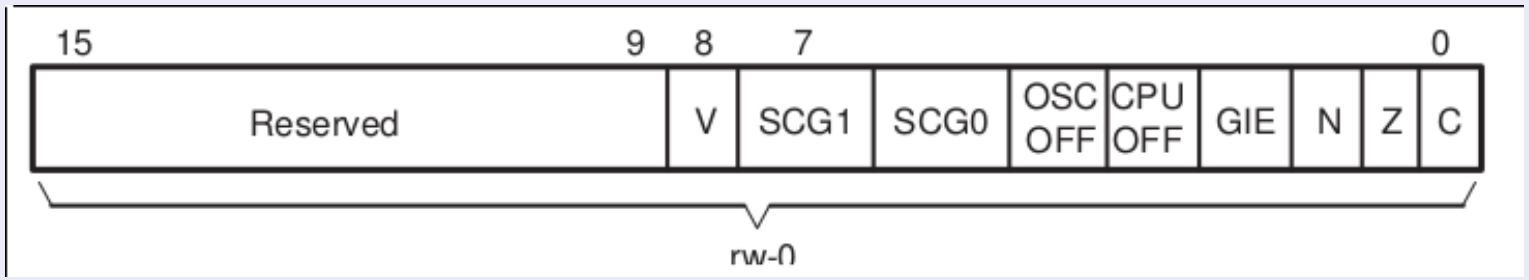
Popping

- Get temporary data back
- Automatically gets data on interrupt/function call return

Registers

R2 Status Register

Contains the Status and Configuration of the processor.



Status

- V flag: Overflow on signed operation (127+1=-128 or -128-1=127)
- SCG0,SCG1,OSCOFF,CPUOFF Clock management
- GIE: Global Interrupt Enable
- N: Negative bit (sign bit of the value)
- Z: Zero bit (result of the last operation is zero)
- C: Carry bit

Registers

R3 Constant generator

Contains frequently used constants depending on the addressing mode. This is transparent to the ASM programmer. It is handled by the assembler.

Constants

Register	As	Constant	Remarks
R2	00	– – – – –	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	–1, word processing

Interrupts

What is an interrupt ?

Change in the program flow to do specific things.

How to do it ?

```
Interrupt_Vector:
do your stuff but keep it short
reti
```

then, add the label into the interrupt vector table

Interrupts

Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up, external reset, watchdog, flash password	WDTIFG KEYV	Reset	0FFFEh	15, highest
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
device-specific			0FFFAh	13
device-specific			0FFF8h	12
device-specific			0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
device-specific			0FFF2h	9
device-specific			0FFF0h	8
device-specific			0FFEEh	7
device-specific			0FFECh	6
device-specific			0FFEAh	5
device-specific			0FFE8h	4
device-specific			0FFE6h	3
device-specific			0FFE4h	2
device-specific			0FFE2h	1
device-specific			0FFE0h	0, lowest

How-to

Move Data

mov src, dest

reads as move source to destination

- Constant to Register mov #0, R7
- Register to Memory mov R7, &ADDRESS
- Memory to Register Indirect mov &ADDRESS, @R7
- Constant to Indirect Autoincrement mov #4, @R7+

Arithmetic operation

add src, dest

read as add source to destination

- Constant to Register add #0, R7
- Register to Memory add R7, &ADDRESS
- Memory to Register Indirect add &ADDRESS, @R7
- Constant to Indirect Autoincrement add #4, @R7+

How-to

Logic Operation

and src, dest

read as and source and destination

- Constant to Register and #0, R7
- Register to Memory and R7, &ADDRESS
- Memory to Register Indirect and &ADDRESS, @R7
- Constant to Indirect Autoincrement and #4, @R7+

Control Flow

- use Compare, or do an arithmetic operation
- use JNE, JEQ, JC, JNC, JZ, JNZ, JGE, JL

Example:

CMP #2, R9

JEQ label

How-to

Call

```
    CALL #label
    Some random stuff
label:
    Do stuff
    ret
```

C structures

If in C

```
if (R6 == 0xBABE) {
    Do something
}
Other code here
```

If in ASM

```
    cmp #0xBABE, R6
    jne other_code
    Do Something
other_code:
    Other code here
```

C structures

For in C

```
for (R6 = 0; R6 < 10; R6 ++) {
    Do something
}
Other code here
```

For in ASM

```
    mov #0, R6
beginning:
    cmp #10, R6
    jge other_code
    Do Something
    inc R6
    jmp beginning
other_code:
    Other code here
```

C structures

While in C

```
while (a == 5) {
    Do something
}
Other things
```

While in ASM

```
begin:
    cmp #5, R6
    jne exit
    Do Something
    jmp begin
exit:
    Other things
```

C structures

Do...While in C

```
do {
    Do something
} while (a == 5);
```

Do...While in ASM

```
begin:
    Do Something
    cmp #5, R7
    jeq begin
```

Watchdog

- Avoid CPU crash
- Needs to be reset before overflow resets the CPU
- Write configuration/Reset at address 0x120

Input/Output

Registers

- PxIN: Input value
- PxOUT: Output value
- PxDIR: Direction value
- PxSEL: Select between IO and Peripheral mode

Timer A

Info

- Does Capture and Compare
- Counts Up/Down, up to Max, or up to defined period TACCR0
- Generates outputs with the Compare

Analog-to-Digital Converter

Info

- Integrated Voltage reference
- Max 200 ksps
- Conversion Synchronized with Timer A

Other Peripherals

Digital

- Analog-to-Digital Converter
- U(S)ART
- Other Timers
- DMA Engine
- Flash Memory Controller
- Multiplier

Analog

- Comparator
- Digital-to-Analog Converter

NOW, ARBEIT UND SCHNELL !!!