# CEG3136 Final Exam Annex Fall 2009

Computer Architecture II (University of Ottawa)

# CEG3136 – Computer Architecture II / CEG3536 Achitecture d'ordinateur II
# Final Exam Annex / Annex d'examen final
# Fall 2009 / Automne 2009

68HCS12 INSTRUCTION LIST (reduced)

**Loads, Stores, and Transfers**

| Function | Mnemonic | IMM | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Clear Memory Byte | CLR | | | X | X | X | | m(ea) <= 0 |
| Clear Accumulator A (B) | CLRA (B) | | | | | | X | A <= 0 |
| Load Accumulator A (B) | LDAA (B) | X | X | X | X | X | | A <= [m(ea)] |
| Load Double Accumulator D | LDD | X | X | X | X | X | | D <= [m(ea, ea+1)] |
| Load Effective Address into SP (X or Y) | LEAS (A,B) | | | | | | | SP <= ea |
| Store Accumulator A (B) | STAA (B) | X | X | X | X | X | | m(ea) <= (A) |
| Store Double Accumulator D | STD | X | X | X | X | X | | m(ea, ea+1) <= D |
| Transfer A to B | TAB | | | | | | X | B <= (A) |
| Transfer A to CCR | TAP | | | | | | X | CCR <= (A) |
| Transfer B to A | TBA | | | | | | X | A <= B |
| Transfer CCR to A | TPA | | | | | | X | A <= (CCR) |
| Exchange D with X (Y) | XGDX | | | | | | X | D <=> (X ) |
| Pull A (B) from Stack | PULA(B) | | | | | | X | A <= [m(SP)], SP <= (SP)+1 |
| Push A (B) onto Stack | PSHA(B) | | | | | | X | SP <= (SP)-1, m(SP) <= A |

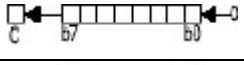
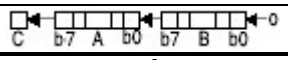
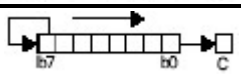
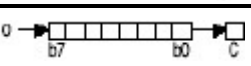
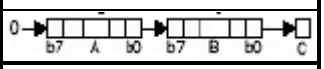
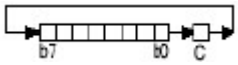
**Arithmetic Operations**

| Function | Mnemonic | IMM | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Add Accumulators | ABA | | | | | | X | A <= (A) + (B) |
| Add with Carry to A (B) | ADCA (B) | X | X | X | X | X | | A <= (A) + [m(ea)]+(C) |
| Add Memory to A (B) | ADDA (B) | X | X | X | X | X | | A <= (A) + [m(ea)] |
| Add Memory to D (16 Bit) | ADDD | X | X | X | X | X | | D <= (D) + [m(ea,ea+1)] |
| Decrement Memory Byte | DEC | | | X | X | X | | m(ea) <= [m(ea)] – 1 |
| Decrement Accumulator A (B) | DECA (B) | | | | | | X | A <= (A) – 1 |
| Increment Memory Byte | INC | | | X | X | X | | m(ea) <= [m(ea)] + 1 |
| Increment Accumulator A (B) | INCA (B) | | | | | | X | A <= (A) + 1 |
| Subtract with Carry from A (B) | SBCA (B) | X | X | X | X | X | | A <= (A) – [m(ea)] – C |
| Subtract Memory from A (B) | SUBA (B) | X | X | X | X | X | | A <= (A) – [m(ea)] |
| Subtract Memory from D (16 Bit) | SUBD | X | X | X | X | X | | D <= (D) – [m(ea,ea+1)] |
| Multiply (byte, unsigned) | MUL | | | | | | X | D <= (A) x (B) |
| Multiply word, unsigned (signed) | EMUL(S) | | | | | | X | Y:D <= (D) x (Y) |
| Unsigned (signed) 32 by 16 divide | EDIV(S) | | | | | | X | X <= (Y:D) /.(X),Y <= quotient, D <= remainder |
| Fractional Divide (D < X) | FDIV | | | | | | X | X <= (D) /.(X), D <= remainder |
| Integer Divide (unsigned) | IDIV | | | | | | X | X <= (D) /.(X), D <= remainder |

**Logical Operations**

| Function | Mnemonic | IMM | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| AND A (B) with Memory | ANDA (B) | X | X | X | X | X | | A <= A • [m(ea)] |
| Bit(s) Test A (B) with Memory | BITA (B) | X | X | X | X | X | | A • [m(ea)] |
| One's Complement Memory Byte | COM | | | X | X | X | | m(ea) <= [/m(ea)] |
| One's Complement A (B) | COMA (B) | | | | | | X | A <= /A |
| OR A (B) with Memory (Exclusive) | EORA (B) | X | X | X | X | X | | A <= A ⊕ [m(ea)] |
| OR A (B) with Memory (Inclusive) | ORAA (B) | X | X | X | X | X | | A <= A + [m(ea)] |

**Shift and Rotate**

| Function | Mnemonic | IMM | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Arithmetic/Logical Shift Left Memory | ASL/LSL | | | X | X | X | |  |
| Arithmetic/Logical Shift Left A (B) | ASLA(B) | | | | | | X | |
| Arithmetic/Logical Shift Left Double | ASLD/LSLD | | | | | | X |  |
| Arithmetic Shift Right Memory | ASR | | | X | X | X | |  |
| Arithmetic Shift Right A (B) | ASRA(B) | | | | | | X | |
| Logical Shift Right A (B) | LSRA(B) | | | | | | X |  |
| Logical Shift Right Memory | LSR | | | X | X | X | | |
| Logical Shift Right D | LSRD | | | | | | X |  |
| Rotate Left Memory | ROL | | | X | X | X | |  |
| Rotate Left A (B) | ROLA(B) | | | | | | X | |
| Rotate Right A (B) | RORA(B) | | | | | | X |  |
| Rotate Right Memory | ROR | | | X | X | X | | |

**Compare & Test**

| Function | Mnemonic | IMM | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Compare A to B | CBA | | | | | | X | (A)-(B) |
| Compare A (B) to Memory | CMPA (B) | X | X | X | X | X | | (A) - [m(ea)] |
| Compare D to Memory (16 Bit) | CPD | X | X | X | X | X | | (D) - [m(ea,ea+1)] |
| Compare SP to Memory (16 Bit) | CPS | X | X | X | X | X | | (SP) - [m(ea,ea+1)] |
| Compare X (Y) to Memory (16 Bit) | CPX | X | X | X | X | X | | (X) - [m(ea,ea+1)] |
| Test memory for 0 or minus | TST | | | X | X | X | | m(ea) - 0 |
| Test A (B) for 0 or minus | TSTA (B) | | | | | | X | (A)-0 |

**Short Branches**

| Function | Mnemonic | REL | DIR | IDX | [IDX] | PC <= ea if |
|---|---|---|---|---|---|---|
| Branch ALWAYS | BRA | X | | | | |
| Branch if Carry Clear | BCC | X | | | | C = 0 ? |
| Branch if Carry Set | BCS | X | | | | C = 1 ? |
| Branch if Equal Zero | BEQ | X | | | | Z = 1 ? |
| Branch if Not Equal | BNE | X | | | | Z = 0 ? |
| Branch if Minus | BMI | X | | | | N = 1 ? |
| Branch if Plus | BPL | X | | | | N = 0 ? |
| Branch if Bit(s) Clear in Memory Byte | BRCLR | | X | X | | [m(ea)]•mask=0 |
| Branch if Bit(s) Set in Memory Byte | BRSET | | X | X | | [/m(ea)]•mask=0 |
| Branch if Overflow Clear | BVC | X | | | | V = 0 ? |
| Branch if Overflow Set | BVS | X | | | | V = 1 ? |
| Branch if Greater Than | BGT | X | | | | Signed > |
| Branch if Greater Than or Equal | BGE | X | | | | Signed ≥ |
| Branch if Less Than or Equal | BLE | X | | | | Signed ≤ |
| Branch if Less Than | BLT | X | | | | Signed < |
| Branch if Higher | BHI | X | | | | Unsigned > |
| Branch if Higher or Same (same as BCC) | BHS | X | | | | Unsigned ≥ |
| Branch if Lower or Same | BLS | X | | | | Unsigned ≤ |
| Branch if Lower (same as BCS) | BLO | X | | | | Unsigned < |
| Branch Never | BRN | X | | | | 3-cycle NOP |

**Long branch** mnemonic = **L** + Short branch mnemonic, e.g.: BRA → **L**BRA

**Loop Primitive Instructions** (counter ctr = A, B, or D)

| Function | Mnemonic | REL | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Decrement counter & branch if =0 | DBEQ | X | | | | | | ctr <= (ctr)-1, if (ctr)=0 => PC <= ea |
| Decrement counter & branch if ≠0 | DBNE | X | | | | | | ctr <= (ctr)-1, if (ctr) ≠0 => PC <= ea |
| Increment counter & branch if =0 | IBEQ | X | | | | | | ctr <= (ctr)+1, if (ctr)=0 => PC <= ea |
| Increment counter & branch if ≠0 | IBNE | X | | | | | | ctr <= (ctr)+1, if (ctr) ≠0 => PC <= ea |
| Test counter & branch if =0 | DBEQ | X | | | | | | if (ctr)=0 => PC <= ea |

**Subroutine Calls and Returns**

| Function | Mnemonic | REL | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Branch to Subroutine | BSR | X | | | | | | SP <= (SP)-2, m(SP) <= (PC), PC <= ea |
| Jump to Subroutine | JSR | | X | X | X | X | | SP <= (SP)-2, m(SP) <= (PC), PC <= ea |
| CALL a Subroutine (expanded memory) | CALL | | X | X | X | X | | SP <= (SP)-2, m(SP) <= (PC), PC <= ea<br>SP <= (SP)-1, m(SP) <= (PPG), PC <= pg |
| Return from Subroutine | RTS | | | | | | X | PC <= [m(SP)], SP <= (SP)+2 |
| Return from call | RTC | | | | | | X | PPG <= [m(SP)], SP <= (SP)+1,<br>PC <= [m(SP)], SP <= (SP)+2 |

| Function | Mnemonic | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|
| Jump | JMP | X | X | X | X | | PC <= ea |

The **jump** instruction allows control to be passed to any address in the 64-Kbyte memory map.

**Stack and Index Register Instructions**

| Function | Mnemonic | IMM | DIR | EXT | IDX | [IDX] | INH | Operation |
|---|---|---|---|---|---|---|---|---|
| Decrement Index Register X (Y) | DEX (Y) | | | | | | X | X <= (X) - 1 |
| Increment Index Register X (Y) | INX (Y) | | | | | | X | X <= (X) + 1 |
| Load Index Register X (Y) | LDX(Y) | X | X | X | X | X | | X <= [m(ea,ea+1)] |
| Pull X (Y) from Stack | PULX | | | | | | X | X <= [m(SP,SP+1)]<br>SP <= (SP) + 2 |
| Push X (Y) onto Stack | PSHX (Y) | | | | | | X | m(SP,SP+1) <= (X)<br>SP <= (SP) - 2 |
| Store Index Register X (Y) | STX (X) | X | X | X | X | X | | m(ea,ea+1) <= X |
| Add Accumulator B to X (Y) | ABX (Y) | | | | | | X | X <= (X) + (B) |
| Decrement Stack Pointer | DES | | | | | | X | SP <= (SP) - 1 |
| Increment Stack Pointer | INS | | | | | | X | SP <= (SP) + 1 |
| Load Stack Pointer | LDS | X | X | X | X | X | | SP <= [m(ea,ea+1)] |
| Store Stack Pointer | STS | X | X | X | X | X | | m(ea,ea+1) <= (SP) |
| Transfer SP to X (Y) | TSX (Y) | | | | | | X | X <= (SP) |
| Transfer X (Y) to SP | TXS (Y) | | | | | | X | SP <= (X) |
| Exchange D with X (Y) | XGDX (Y) | | | | | | X | (D) <=> (X) |

| Function | Mnemonic | INH | Operation |
|---|---|---|---|
| Return from Interrupt | RTI | X | $(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$<br>$(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow S$<br>$(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow S$<br>$(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow$<br>$(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow S$ |
| Software Interrupt | SWI | X | |
| Wait for Interrupt | WAI | X | |

**Interrupt Handling**

The software interrupt (SWI) instruction is similar to a JSR instruction, except the contents of all working CPU registers are saved on the stack rather than just the return address. SWI is unusual in that it is requested by the software program as opposed to other interrupts that are requested asynchronously to the executing program.

# Register and Bit Definitions for 9S12DG256 Peripheral Modules

## DATA REGISTERS

### PARALLEL PORTS

| PORTA | $0000 | PORTB | $0001 | PORTE | $0008 | PORTK | $32 |
|---|---|---|---|---|---|---|---|
| PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 |
| PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
| PE7 | PE6 | PE5 | PE4 | PE3 | PE2 | PE1 | PE0 |
| PK7 | Pk6 | PK5 | PK4 | PK3 | PK2 | PK1 | PK0 |

### TIMER

**TCNT** $0044 $0045 **Timer Count Register**

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

**TC0** $0050 $0051 **Timer Input-Capture/ Output-Compare Register 0**

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

**TCx** **x=0…7** $0050+2*x

### SERIAL INTERFACE-SCI

**SC0DRH** $00CE **SCI Data Register High**

| R8 | T8 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**SC0DRL** $00CF **SCI Data Register Low**

| R7T7 | R6T6 | R5T5 | R4T4 | R3T3 | R2T2 | R1T1 | R0T0 |
|---|---|---|---|---|---|---|---|

### SERIAL INTERFACE-SPI

**SP0DR** $00Dd **SPI Data Register**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|

### ANALOG-TO-DIGITAL SUBSYSTEM

**ADRxH** $90+2x **ATD Result Registers High**

| Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 |
|---|---|---|---|---|---|---|---|

**ADRxL**: $91+2x **ATD Result Registers Low**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|

**x=0…7**

## CONTROL REGISTERS

**DDRA** $0002 **DDRB** $0003 **DDRE** $0009 **DDRK** $0033

| DDRA7 | DDRA6 | DDRA5 | DDRA4 | DDRA3 | DDRA2 | DDRA1 | DDRA0 |
|---|---|---|---|---|---|---|---|
| DDRB7 | DDRB6 | DDRB5 | DDRB4 | DDRB3 | DDRB2 | DDRB1 | DDRB0 |
| DDRE7 | DDRE6 | DDRE5 | DDRE4 | DDRE3 | DDRE2 | DDRE1 | DDRE0 |
| DDRK7 | DDRK6 | DDRK5 | DDRK4 | DDRK3 | DDRK2 | DDRK1 | DDRK0 |

**PUCR** $000C

| PUPKE | 0 | 0 | PUPEE | 0 | 0 | PUPBE | PUPAE |
|---|---|---|---|---|---|---|---|

**TIOS** $0040 **Timer Input-Capture (IOS=0)/Output-Compare (IOS=1) Select Register**

| IOS7 | IOS6 | IOS5 | IOS4 | IOS3 | IOS2 | IOS1 | IOS0 |
|---|---|---|---|---|---|---|---|

**CFORC** $0041 **Timer Compare Force Register**

| FOC7 | FOC6 | FOC5 | FOC4 | FOC3 | FOC2 | FOC1 | FOC0 |
|---|---|---|---|---|---|---|---|

**OC7M** $0042 **Timer Output Compare 7 Mask**

| OC7M7 | OC7M6 | OC7M5 | OC7M4 | OC7M3 | OC7M2 | OC7M1 | OC7M0 |
|---|---|---|---|---|---|---|---|

**OC7D** $0043 **Timer Ouput Compare 7 Data Register**

| OC7D7 | OC7D6 | OC7D5 | OC7D4 | OC7D3 | OC7D2 | OC7D1 | OC7D0 |
|---|---|---|---|---|---|---|---|

**TCTL1/TCTL2** $0048/49 **Timer Control Register 1/2**

| OM7 | OL7 | OM6 | OL6 | OM5 | OL5 | OM4 | OL4 |
|---|---|---|---|---|---|---|---|
| OM3 | OL3 | OM2 | OL2 | OM1 | OL1 | OM0 | OL0 |

**TCTL3/TCTL4** $004A/4B **Timer Control Register 4**

| EDG7B | EDG7A | EDG6B | EDG6A | EDG5B | EDG5A | EDG4B | EDG4A |
|---|---|---|---|---|---|---|---|
| EDG3B | EDG3A | EDG2B | EDG2A | EDG1B | EDG1A | EDG0B | EDG0A |

**TSCR1** $0046 **Timer System Control Register 1**

| TEN | TSWAI | TSFRZ | TFFCA | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**TSCR2** $004D **Timer System Control Register 1**

| TOI | 0 | 0 | 0 | TCRE | PR2 | PR1 | PR0 |
|---|---|---|---|---|---|---|---|

**TIE** $004C **Timer Interrupt Enable Register**

| C7I | C6I | C5I | C4I | C3I | C2I | C1I | C0I |
|---|---|---|---|---|---|---|---|

**SC0BDH,** $00C8 $00C9 **SCI Baud Rate Control Register**

| 0 | 0 | 0 | SBR12 | SBR11 | SBR10 | SBR9 | SBR8 |
|---|---|---|---|---|---|---|---|
| SBR7 | SBR6 | SBR5 | SBR4 | SBR3 | SBR2 | SBR1 | SBR0 |

**SC0CR1** $00CA **SCI Control Register 1**

| LOOPS | SCISWAI | RSRC | **M** | WAKE | ILT | **PE** | **PT** |
|---|---|---|---|---|---|---|---|

**SC0CR2** $00CB **SCI Control Register 2**

| TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK |
|---|---|---|---|---|---|---|---|

**SP0BR** $00DA **SPI Baud Rate Control Register**

| 0 | SPPR2 | SPPR1 | SPPR1 | 0 | SPR2 | SPR1 | SPR0 |
|---|---|---|---|---|---|---|---|

**SP0CR1** $00D8 **SPI Control Register 1**

| SPIE | SPE | SPTIE | MSTR | CPOL | CPHA | SSOE | LSBF |
|---|---|---|---|---|---|---|---|

**SP0CR2** $00D9 **SPI Control Register 2**

| 0 | 0 | 0 | MODFEN | BIDRIOE | 0 | SPISWAI | SPC0 |
|---|---|---|---|---|---|---|---|

**ATDCTL2** $82 **ATD Control Register 2**

| ADPU | AFFC | AWAI | ETRIGLE | ETRIGP | ETRIG | ASCIE | ASCIF |
|---|---|---|---|---|---|---|---|

**ATDCTL3** $83 **ATD Control Register 3**

| 0 | S8C | S4C | S2C | S1C | FIFO | FRZ1 | FRZ2 |
|---|---|---|---|---|---|---|---|

**ATDCTL4** $84 **ATD Control Register 4**

| SRES8 | SMP1 | SMP0 | PRS4 | PRS3 | PRS2 | PRS1 | PRS0 |
|---|---|---|---|---|---|---|---|

**ATDCTL5** $85 **ATD Control Register 5**

| DJM | DSGN | SCAN | MULT | 0 | CC | CB | CA |
|---|---|---|---|---|---|---|---|

**INTCR** $001E **Interrupt Control Register**

| IRQE | IRQEN | DLY | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

## STATUS REGISTERS

**TFLG1** $004E **Timer Interrupt Flag Register 1**

| C7F | C6F | C5F | C4F | C3F | C2F | C1F | C0F |
|---|---|---|---|---|---|---|---|

***TFLG2*** ***$004F*** ***Timer Interrupt Flag 2***

| TOF | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

| OMn | OLn | OCn out |
|---|---|---|
| 0 | 0 | Disconnected |
| 0 | 1 | Toggle |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| EDGnB | EDGnA | ICn in |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | **Rising** |
| 1 | 0 | **Falling** |
| 1 | 1 | **Any** |

**PR = 0…5;**
**prescale factor = $2^{PR}$**

| PR2 | PR1 | PR0 | Fact |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 4 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 16 |
| 1 | 0 | 1 | 32 |
| 1 | 1 | 0 | 64 |
| 1 | 1 | 1 | 128 |

| Baud Rate | SER |
|---|---|
| 2400 | 651 |
| 9600 | 163 |
| 19,200 | 81 |
| 38,400 | 41 |

**PT=0** => even
**M=0** =>· 1 start bit, 8 data bits, 1 stop bit
**RDRF = 0** => SC0DR is empty
**$FFD6 = SCI0 interrupt vector address**

**SC0SR1** $00CC **SCI Status Register 1**

| **TDRE** | TC | **RDRF** | IDLE | OR | NF | FE | **PF** |
|---|---|---|---|---|---|---|---|

**SC0SR2** $00CD **SCI Status Register 2**

| **0** | 0 | 0 | 0 | 0 | BRK13 | TXDIR | RAF |
|---|---|---|---|---|---|---|---|

**E-Clock Divisor = ((SPPR+1)*2)^SPR for SPPR > 0**

**SP0SR** $00DB **SPI Status Register**

| SPIF | 0 | SPTEF | MODF | 0 | 0 | 0 | **0** |
|---|---|---|---|---|---|---|---|

**ATDSTAT0** $86 **ATD Status Register 0**

| **SCF** | 0 | ETORF | FIFOR | 0 | CC2 | CC1 | CC0 |
|---|---|---|---|---|---|---|---|

**ATDSTAT1** $86 **ATD Status Register 1**

| CCF7 | CCF6 | CCF5 | CCF4 | CCF3 | CCF2 | CCF1 | CCF0 |
|---|---|---|---|---|---|---|---|

SampleDuration = 2^(SMP+1) ATD clock

**HPRIO** $001F **Highest Priority I Interrupt Register**

| PSEL7 | PSEL6 | PSEL5 | PSEL4 | PSEL3 | PSEL2 | PSEL1 | 0 |
|---|---|---|---|---|---|---|---|

Timer Module

- TEN - Activate (=1) and deactivate (=0) timer
- AFFC - allows automatic clear of flags (=1)
- TOI enable (=1) overflow interrupt (from $FFFF to $0000) of the TCNT
- PR2, PR1, PR0 prescale bits (= $2^{PR}$) applied to system clock for incrementing timer
- IOS0 to IOS7 – Configure channel as output compare (=1) or input-capture (=0)
- FOC0-FOC1 – Force output-compare event on channel
- OC7M0-OC7M7 – Allow channel 7 to affect channel pin (=1)
- OC7D0-OC7D7 – Value of level for channel 7 to output to channel pin
- C0I-C7I – Enable a channel interrupt
- C0F-C7F – Channel flag set when event occurs
- TOF – Counter Overflow flag

ATD Module

- ADPU enable (=1) converter
- AFFC enable (=1) fast clear
- ASCIE enable (=1) interruption at the completion of a conversion sequence
- ASCFI flags (=1) a sequence conversion
- S8C,S4C,S2C,S1C defines number of conversions per sequence (when set to 0, or S8C=1,8 conversions in a sequence)
- FIFO – Use FIFO discipline to fill registers when set to 1 (otherwise fill in starting at register ADR0).
- SRES8 selects resolution (1 = 8 bits, 0 = 10 bits)
- SMP1, SMP2 – sample time (= $2^{SMP+1}$)
- PRS4, PRS3,PRS2,PRS1,PRS0 – prescaler of system bus clock (of frequency BUS = typically 24 MHz) to generate ATD clock (of frequency CC):
  CC = (BUS/(2*(PRS+1))), where PRS is value represented by PRS4 to PRS0.
- DJM – Right justify (=1) or left justify (=0) results
- DSGN – Used signed (=1) or unsigned numbers (=0) (significant only if DJM=0)
- SCAN – single conversion (=0) or scan mode (=1)
- MULT – single pin (=0) or multiple pins (=1)
- CC, CB, CA – Selects pin for single conversion or first pin in the case of multiple conversion.

ATD Cycles per conversion
2 (initial sample time) + 2 to 16 (final sample time) + 8 or 10 (conversion time for 8 bits or 10 bits).

Serial Interface Module – SCI subsystem

- M: Two modes 8 data bits or 9 data bits (with 1 start bit and 1 stop bit)
- PE, PT: Enable parity (PE) and type of parity (PT)
- TIE, TCIE, RIE: Bits for enabling interruptions for reception and transmission
- TE, RE: Enable transmission (TE) and reception (RE)
- TDRE: 1 indicates that more data can be written into the data register
- TC: 1 indicates that transmission of data is complete
- RDRF: 1 indicates that reception data register is full
- OR, NF, FE, PF: error bits, equal 1 when an error is detected during reception (overrun, noise, framing, parity)
- RAF: 1 indicates that reception of a character is underway.
- MODF – Set when SS configured in Master for error detection (input)

# Port H

**Address Offset:$__20**

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Read:<br>Write: | PTH7 | PTH6 | PTH5 | PTH4 | PTH3 | PTH2 | PTH1 | PTH0 |
| SPI: | $\overline{SS2}$ | SCK2 | MOSI2 | MISO2 | $\overline{SS1}$ | SCK1 | MOSI1 | MISO1 |
| Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

= Reserved or unimplemented

**Figure 3-30 Port H I/O Register (PTH)**

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.

The SPI function takes precedence over the general purpose I/O function associated with if enabled. *Refer to SPI Block Guide for details.*

**Address Offset: $__22**

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Read:<br>Write: | DDRH7 | DDRH6 | DDRH5 | DDRH4 | DDRH3 | DDRH2 | DDRH1 | DDRH0 |
| Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

= Reserved or unimplemented

**Figure 3-32 Port H Data Direction Register (DDRH)**

Read:Anytime.

Write:Anytime.

This register configures each port H pin as either input or output.

DDRH[7:0] — Data Direction Port H

1 = Associated pin is configured as output.

0 = Associated pin is configured as input.

```
/**************** interrupt vector numbers ****************/
#define VectorNumber_Vatd1              23
#define VectorNumber_Vatd0              22
#define VectorNumber_Vsci1              21
#define VectorNumber_Vsci0              20
#define VectorNumber_Vtimovf            16
#define VectorNumber_Vtimch7            15
#define VectorNumber_Vtimch6            14
#define VectorNumber_Vtimch5            13
#define VectorNumber_Vtimch4            12
#define VectorNumber_Vtimch3            11
#define VectorNumber_Vtimch2            10
#define VectorNumber_Vtimch1            9
#define VectorNumber_Vtimch0            8
#define VectorNumber_Virq               6
#define VectorNumber_Vxirq              5
#define VectorNumber_Vswi               4

// Peripheral registers
#define PORTA  _PORTAB.Overlap_STR.PORTASTR.Byte // PORT A Data Register
#define DDRA   _DDRAB.Overlap_STR.DDRASTR.Byte  // PORT A Data Direction Register
#define PORTB  _PORTAB.Overlap_STR.PORTBSTR.Byte // PORT B Data Register
#define DDRB   _DDRAB.Overlap_STR.DDRBSTR.Byte  // PORT B Data Direction Register
#define PORTK  _PORTK.Byte      // PORT K Data Register
#define DDRK   _DDRK.Byte       // PORT K Data Direction Register
#define PUCR   _PUCR.Byte       // Pullup Control Register
#define INTCR  _INTCR.Byte      // Interrupt Control Register
#define PTH    _PTH.Byte        // Port H Data Register
#define DDRH   _DDRH.Byte       // Port H Data Direction Register

// Timer Registers
#define TSCR1  _TSCR1.Byte      // Timer System Control Register 1
#define TSCR2  _TSCR2.Byte      // Timer System Control Register 2
#define TIOS   _TIOS.Byte       // Timer Input Capture/Output Compare Select
#define CFORC  _CFORC.Byte      // Timer Compare Force Register
#define OC7M   _OC7M.Byte       // Output Compare 7 Mask Register
#define OC7D   _OC7D.Byte       // Output Compare 7 Data Register
#define TCNT   _TCNT.Word       // Timer Count Register
#define TCTL1  _TCTL1.Byte      // Timer Control Register 1
#define TCTL2  _TCTL2.Byte      // Timer Control Register 2
#define TCTL3  _TCTL3.Byte      // Timer Control Register 3
#define TCTL4  _TCTL4.Byte      // Timer Control Register 4
#define TIE    _TIE.Byte        // Timer Interrupt Enable Register;
#define TFLG1  _TFLG1.Byte      // Main Timer Interrupt Flag 1
#define TFLG2  _TFLG2.Byte      // Main Timer Interrupt Flag 2
#define TC0    _TC0.Word        // Timer Data Register 0
#define TC1    _TC1.Word        // Timer Data Register 1
#define TC2    _TC2.Word        // Timer Data Register 2
#define TC3    _TC3.Word        // Timer Data Register 3
#define TC4    _TC4.Word        // Timer Data Register 4
#define TC5    _TC5.Word        // Timer Data Register 5
#define TC6    _TC6.Word        // Timer Data Register 6
#define TC7    _TC7.Word        // Timer Data Register 7
```