V65C816 General Purpose 16 Bit Microprocessor Datasheet

Design by: Valerio Venturi (valerioventuri@gmail.com)

The V65C816 is a VHDL RTL softcore 100% software compatible with the original silicon WDC65C816 CPU but with some new instructions:

- two fast multiply 16X16-32 bit instructions
- multitasking context save and restore fast instructions
- two fast save and restore AXY instruction
- register exchange instructions
- Improved execution time for some instructions and addressing modes.
- All the two bytes unused opcodes are treated as NOPs.

New Opcodes:

PHR (0x42/0x8B) **P**us**H** **R**egisters push **C,X,Y** to stack (flags: unaffected)
PLR (0x42/0xAB) **P**ul**L** **R**egisters **C,X,Y** in reversed order from stack (flags: unaffected)
SAV (0x42/0x90) (**SAVE**) push **C,X,Y,P,PBR,D** to stack (flags: unaffected)
RST (0x42/0x91) (**RESTORE**) pull **C,X,Y,P,PBR,D** in reversed order from stack (flags: all)
MPU (0x42/0x8E) **M**ulti**P**ly **U**nsigned 16X16->32 bit (flags: **Z**)
MPS (0x42/0x8F) **M**ulti**P**ly **S**igned 15X15->31 bit (with sign) (flags: **NZ**)
XYX (0x42/0xEB) e**X**change **Y** and **X** (flags: unaffected)
XAX (0x42/0x0B) e**X**change **A** and **X** (flags: unaffected)
XAY (0x42/0x2B) e**X**change **A** and **Y** (flags: unaffected)
EXT (0x42/0xEC) **EXT**ends sign of accumulator **A** to **B**
NEG (0x42/0xED) **NEG**ates contents of accumulator

NOTE: all new instructions listed above must be preceded by a WDM (0x42) opcode

The WDM opcode in the 65C816 WDC was intended as a prefix for new opcodes since the CPU used all 256 opcodes (included WDM), but it was never used because the WDC never made a 65C816 with new instructions, hence the **WDM** opcode in the silicon version it is currently only a **NOP**.
In my version of the 65C816 all the instructions are treated with 9 bit length instead of 8 bit, the **WDM** opcode simply sets the 9th bit to '1' to distinguish the next opcode as new instruction instead of one of the old ones.
But in addition to this **WDM** activates a non-interruptible (atomic) sequence that ends with the end of the execution of the new instruction, therefore during a **WDM->New Opcode** sequence all interrupts are masked (NMI included), this to guarantee that the execution of the new instruction is incorruptible by other processes (such as interrupts mentioned above).
Example:

```
        WDM       ;now all interrupts are masked (IRQ and NMI)
        PHR       ;interrupt mask ends at end of execution of this instruction
        NOP       ;now interrupts are enabled (IRQ obviously if it was enabled before)
```

For what has been said it is therefore very important that the **WDM** instruction is followed by a new operating code (among those defined) otherwise the behavior of the CPU is unpredictable.

Description of new instructions.

PHR

Native mode:
Pushes **C,X,Y** registers to stack, the size is always 16 bit regardless of status of MX flags.

```
        WDM     ;new opcode prefix
        PHR     ;pushes C,X,Y to stack (all registers are always saved as 16 bit size)
```

replaces:

```
        PHA
        PHX
        PHY
```

(assuming both **MX** flags are zero)

Emulation mode:
Pushes **B,A,X,Y** registers to stack, the size is always 8 bit
Example in emulation mode mode:

```
        WDM     ;new opcode prefix
        PHR     ;pushes B,A,X,Y
```

replaces:

```
        XBA
        PHA     ;saves B
        XBA
        PHA     ;saves A
        PHX
        PHY
```

PLR

Native mode:
Pulls **C,X,Y** registers from stack, the size is always 16 bit regardless of status of MX flags.

```
        WDM     ;new opcode prefix
        PLR     ;pulls C,X,Y from stack (all registers are always pulled as 16 bit size)
```

replaces:

```
        PLY     ;pulls Y
        PLX     ;pulls X
        PLA     ;pulls C
```

(assuming both **MX** flags are zero)

Emulation mode:
Pulls **B,A,X,Y** registers, the size is always 8 bit.

```
        WDM     ;new opcode prefix
        PLR     ;pulls B,A,X,Y
```

replaces:

```
        PLY     ;pulls Y
        PLX     ;pulls X
        PLA     ;pulls A
        XBA
        PLA     ;pulls B
        XBA
```

Purpose of **PHR/PLR** instructions is to save and restore **A,X,Y** registers with only a instruction instead three or more.
The **PHR/PLR** instructions don't affect the **P** register.

SAV

Native mode:
Pushes **C,X,Y,P,PBR,D** registers to stack, the size of **C,X,Y** is always 16 bit regardless of status of **MX** flags.

```
        WDM     ;new opcode prefix
        SAV     ;pushes C,X,Y,P,PBR,D to stack (the registers C,X,Y are always saved as 16 bit size)
```

replaces:

```
        PHA
        PHX
        PHY
        PHP
        PHB
        PHD
```

assuming both **MX** flags are zero

Note: in native mode the instruction pushes **P** register to stack in order to remember the size of **X,Y** registers (**X** flag), we must remember that the MSB portion of **X,Y** registers is always forced to zero if **X** flag is set to '1'.
The **RST** instruction when pulls **P** from stack restores the **MX** flags before pull **C,X,Y** from stack.

Emulation mode:
Pushes **B,A,X,Y,PBR,D** registers to stack, the size is always 8 bit

```
        WDM       ;new opcode prefix
        PHR       ;pushes B,A,X,Y,PBR,D
```

replaces:

```
        XBA
        PHA       ;pushes B
        XBA
        PHA       ;pushes A
        PHX       ;pushes X
        PHY       ;pushes Y
        PHB       ;pushes PBR
        PHD       ;pushes D
```

Note: in emulation mode the instruction don't pushes **P** register to stack because the **XY** register are always forced to 8 bit size.

RST

Native mode:
Pulls **C,X,Y,P,PBR,D** registers from stack, the size of **C,X,Y** is always 16 bit regardless of status of **MX** flags.

```
        WDM       ;new opcode prefix
        RST       ;pulls C,X,Y,P,PBR,D from stack (registers C,X,Y are always pulled as 16 bit size)
```

replaces:

```
        PLD       ;pulls D
        PLB       ;pulls B
        PLP       ;pulls P
        PLY       ;pulls Y
        PLX       ;pulls X
        PLA       ;pulls A
```

(assuming both **MX** flags are zero)

Emulation mode:
Pulls **B,A,X,Y,PBR,D** registers, the size of **XY** is always 8 bit.

```
        WDM        ;new opcode prefix
        RST        ;B,A,X,Y,PBR,D
```

replaces:

```
        PLD       ;pulls D
        PLB       ;pulls B
        PLY       ;pulls Y
        PLX       ;pulls X
        PLA       ;pulls A
        XBA
        PLA       ;pulls B
        XBA
```

Purpose of **SAV/RST** instructions is to save and restore all registers with only a instruction instead six or more, this is very useful in situations where a quick context switch is needed, for example in a preemptive (interrupt-based) multitasking system.
The **SAV** instructions don't affect the **P** register, **RST** instead modifies all the flags of **P** (only when

in native mode).

MPU

This instruction performs a 16X16 bit unsigned multiply and returns a 32 bit result.
The register **C** and **X** must be loaded with the two factors and the multiply returns the 32 bit result in **C** (LSB) and **X** (MSB).

Example, we want to multiply 1234 by 100:

```
            REP %00110000       ;set registers A,X to 16 bit size
            LDA factor1         ;loads factor1 to A
            LDX factor2         ;loads factor2 to X
            WDM                 ;new opcode prefix
            MPU                 ;multiply unsigned
            STA lsb_result      ;save LSB result
            STX msb_result      ;save MSB result

factor1    .word 1234
factor2    .word 100

lsb_result .word 0
msb_result .word 0
```

Note: the hardware multiplication is based on a loop which at each clock shifts the value contained in the **X** register to the right and ends when the value is zero, therefore to speed up the execution of the instruction it is better to load the smallest factor on the register **X**, of course if you know it.

MPS

This instruction performs a 15X15 bit signed multiply and returns a 31 bit result.
The register **C** and **X** must be loaded with the two factors and the multiply returns the 31 bit signed result in **C** (LSB) and **X** (MSB).

Example, we want to multiply 1234 by -100:

```
            REP %00110000       ;set registers A,X to 16 bit size
            LDA factor1         ;loads factor1 to A
            LDX factor2         ;loads factor2 to X
            WDM                 ;new opcode prefix
            MPS                 ;multiply signed
            STA lsb_result      ;save LSB result
            STX msb_result      ;save MSB result

factor1    .word 1234
factor2    .word -100

lsb_result .word 0
msb_result .word 0
```

Note: the hardware multiplication is based on a loop which at each clock shifts the value contained in the **X** register to the right and ends when the value is zero, therefore to speed up the execution of the instruction it is better to load the smallest factor on the register **X**, of course if you know it.

Example of preemptive multitasking with V65C816

```
;this simple example is a preemptive multistasking for V65C816 for four tasks but could be extended
for more task if needed.


;RAM locations
ctask               .byte  1    ;current task index
ntask               .byte  1    ;number of defined tasks
ststsk              .byte  4    ;status of task #0-3 (enable or disabled)
sptsk               .byte  8    ;16 bit stack pointers of task #0-3


;we arrive here after reset
start               sei                 ;masks IRQ interrupt
                    clc
                    xce                 ;set native mode
                    sep    %00110000    ;AXY=8 bit
                    MX     %11          ;tell to assembler AXY = 8 bit
                    lda    ssp+1        ;msb sp --> a
                    xba
                    lda    ssp          ;lsb sp --> a
                    tcs                 ;a --> sp
                    stz    ready        ;reset ready flag

                    ;TASK INITIALIZATION
                    lda   numtsk        ;initializes number of task
                    sta   ntask
                    ldy   #0
                    tyx
                    sty   tmp1

res6                lda   ssp+1,y      ;msb sp --> a
                    xba
                    lda   ssp,y        ;lsb sp --> a
                    tcs                ;a --> sp
                    lda   pctask+1,y   ;msb pc task --> a
                    pha                ;a --> stack
                    lda   pctask,y     ;lsb pc task --> a
                    pha                ;a --> stack
                    ldx   tmp1
                    lda   #%00110100   ;0 --> a  (task will be start with interrupt (IRQ) disabled,
ALU in binary mode and AXY = 8 bit)
                    pha                ;p --> stack
                    lda   emtask,x     ;check if task is emulation or native mode
                    sta   emtsk,x      ;saves the emulation/native mode flag for this task
                    beq   res7         ;if native mode
                    lda   #0           ;0 --> a       (emulation mode)
                    pha                ;a --> stack
                    pha                ;a --> stack
                    pha                ;x --> stack
                    pha                ;x --> stack
                    bra   res8

res7                lda   #0           ;0 --> a       (native mode)
                    pha                ;a --> stack
                    pha                ;a --> stack
                    pha                ;x --> stack
                    pha                ;x --> stack
                    pha                ;y --> stack
                    pha                ;y --> stack
                    lda   #%00110000   ;task starts always in AXY = 8 bit
                    pha                ;p -> stack

res8                lda   #0
                    pha                ;b -> stack
                    pha                ;d -> stack
                    pha
res9                tsc                ;sp --> a
                    sta   sptsk,y      ;lsb sp --> sptsk
                    xba
                    sta   sptsk+1,y    ;msb sp --> sptsk
                    ldx   tmp1
                    lda   stasts,x     ;status task --> a
                    sta   ststsk,x     ;a --> ststsk+y
                    iny                ;y += 2
                    iny
                    inx
                    stx   tmp1
```

```
                    cpx   numtsk       ;x < numtsk ?
                    bcc   res6         ;if yes repeat loop
                    sep   %00110000
                    lda   ssp+1        ;msb sp --> a
                                                xba
                    lda   ssp          ;lsb sp --> a
                    tcs                ;a --> sp
                    stz   ctask        ;0 --> ctask (we start with task #0)
                    ;HERE DO SOMETHING TO START A IRQ INTERRUPT GENERATED BY A FREERUN TIMER (ES: 1-
10 KHZ FREQ)

                    brl   task0        ;start task #0
                    nop
                    nop


;task #0:
task0               cli                      ;enables IRQ
task0_loop          nop                      ;do something...
                    brl   task0_loop         ;in this implementation tasks MUST BE a while loop

;task #1:
task1               cli                      ;enables IRQ
task1_loop          nop                      ;do something...
                    brl   task1_loop         ;in this implementation tasks MUST BE a while loop

;task #2:
task2               cli                      ;enables IRQ
task2_loop          nop                      ;do something...
                    brl   task2_loop         ;in this implementation tasks MUST BE a while loop

;task #3:
task3               cli                      ;enables IRQ
task3_loop          nop                      ;do something...
                    brl   task3_loop         ;in this implementation tasks MUST BE a while loop



;IRQ interrupt (task switcher)
irq                 wdm                ;new opcode prefix
                    sav                ;save a,x,y,p,dbr,d on stack
                    clc                ;set native mode
                    xce
                    sep   %00110000    ;a-x-y = 8 bit
                    MX    %11          ;tell to assembler AXY = 8 bit

                    ror   a            ;carry --> bit 7 a (bit E --> bit 7 a)
irq1                ldx   ctask        ;current task index --> y
                    sta   emtsk,x      ;saves emulation/native mode of interrupted task
                    txa
                    asl   a            ;multiply * 2
                    tay
                    tsc                ;sp --> a
                    sta   sptsk,y      ;save lsb sp
                    xba
                    sta   sptsk+1,y    ;save msb sp
                    ldy   ctask        ;y = current task
irq2                iny                ;y = y +1 (point to next task)
                    cpy   ntask        ;reached end of number of task
                    bcc   irq3         ;if no
                    ldy   #0           ;0 --> y (restart scan of task list)
irq3                sty   ctask        ;y --> ctask (current task)
                    tya                ;test y
                    tax                ;copy to x
                    beq   irq4         ;if task #0
                    lda   ststsk,x     ;the task to restart is enabled ?
                    beq   irq2         ;if no repeat the scan of task list
irq4                txa
                    asl   a            ;multiply * 2
                    tay
                    lda   sptsk+1,y    ;get the sp of next task (msb)
                    xba
                    lda   sptsk,y      ;(lsb) sp
                    tcs                ;initializes sp of next task
                    ldx   ctask        ;x is index of next task
                    lda   emtsk,x      ;read the emulation/native mode of next task
                    rol   a            ;transfer to C flag
                    bit   timer1_cli   ;clear timer interrupt
                    xce                ;restore emulation/native mode of the next task
                    wdm                ;new opcode prefix
```

```
                rst             ;restore a,x,y,p,dbr,d from stack
                rti             ;returns from interrupt (and restart the next task)


pctask          .word  task0,task1,task2,task3     ;pointers of start PC tasks
ssp             .word  $02ff,$03ff,$04ff,$05ff     ;top of stack pointer of tasks
stasts          .byte  $ff,$ff,$ff,$ff             ;status of tasks, all are enabled
emtask          .byte  $00,$00,$00,$80             ;emulation/native mode of tasks
                                                   ;$00=native mode; $80 =emulation mode
numtsk          .byte  4                           ;number of defined tasks
```