D16 Processor Reference Manual

Michael Nolan

Aug 16, 2016

Contents

1 The Processor **1**
- 1.1 General Purpose Registers . . . 1
- 1.2 Flags . . . 1

2 Instruction Set **1**
- 2.1 ADD . . . 1
- 2.2 SUB . . . 1
- 2.3 PUSH . . . 2
- 2.4 POP . . . 2
- 2.5 MOV . . . 2
  - 2.5.1 general MOV encoding . . . 2
  - 2.5.2 special byte MOV . . . 2
- 2.6 AND . . . 2
- 2.7 OR . . . 3
- 2.8 XOR . . . 3
- 2.9 NOT . . . 3
- 2.10 NEG . . . 3
- 2.11 LD . . . 3
- 2.12 ST . . . 4
- 2.13 CMP . . . 4
- 2.14 JMP . . . 4
- 2.15 CALL . . . 4
- 2.16 RET . . . 4
- 2.17 SHL . . . 5
- 2.18 SHR . . . 5
- 2.19 ROL . . . 5
- 2.20 RCL . . . 5
- 2.21 LDCP . . . 5
- 2.22 ADC . . . 5
- 2.23 SBB . . . 6
- 2.24 SET . . . 6
- 2.25 TEST . . . 6
- 2.26 PUSHLR . . . 6
- 2.27 SAR . . . 6

3 Condition Codes **7**

4 Coprocessor Registers **7**

5 Peripherals **8**
- 5.1 Peripheral registers . . . 8
- 5.2 LEDs . . . 8
  - 5.2.1 IO_LED_DATA . . . 8
- 5.3 UART . . . 8
  - 5.3.1 IO_UART_DATA . . . 8
  - 5.3.2 IO_UART_STATUS . . . 8
  - 5.3.3 IO_UART_BAUD . . . 9
- 5.4 Timer . . . 9
  - 5.4.1 IO_TIMER_DATA . . . 9
  - 5.4.2 Usage . . . 9
- 5.5 Sound . . . 9
  - 5.5.1 IO_SOUND_DATA . . . 9
  - 5.5.2 Usage . . . 9

5.6 DMA 9
5.6.1 DMA_CONTROL 9
5.6.2 DMA_LOCAL_ADDRESS 9
5.6.3 DMA_REMOTE_ADDRESS 10
5.6.4 DMA_COUNT_ADDRESS 10
5.6.5 DMA Types 10
5.6.6 Usage 10

1 The Processor

The D16 Processor is a very simple, RISC like 16 bit processor with variable length instructions. It has 8 general purpose registers, 32 special purpose registers, and support for up to 64K of memory.

1.1 General Purpose Registers

The D16 processor defines 8 general purpose registers, called r0 - r7. 2 of these, although they behave the same as the other registers, have special meaning to the processor and in the ABI, and they are as follows:

r6: This is generally used as the pointer to the start of a stack frame, but has no special meaning to the processor

r7: This is the stack pointer, and is manipulated via the stack instructions (push and pop)

1.2 Flags

The processor also contains several flags in Special Register 0.

Zero	set if the result of the last computation is 0
Sign	set if the result is negative (bit 15 is set)
Carry	set if there was a carry or borrow in the past computation
oVerflow	set if there was a signed overflow in the last computation

2 Instruction Set

Most instructions come in 2 formats, register and immediate. The immediate versions of an instruction will have bit 7 set in the opcode field and the 16 bit immediate in the word following the instruction. In the subsequent definitions, op2 will refer to the immediate value if the instruction has an immediate, otherwise it refers to rS.

2.1 ADD

Immediate	opcode	Reserved	source	dest
Imm	000001	00	rS	rD

`ADD rD, <rS or immediate>`

rD = rD + op2
Updates flags

2.2 SUB

Immediate	opcode	Reserved	source	dest
Imm	000010	00	rS	rD

`SUB rD, <rS or immediate>`

rD = rD - op2
Updates flags

2.3 PUSH

Immediate	opcode	Reserved	source	dest
Imm	000011	00	∼rS	rD

```
PUSH <rD or immediate>[, rS]
```

rS = rS - 2
memory[rS] = rD
If no rS is specified, rS defaults to r7 This instruction does not update the flags

2.4 POP

Immediate	opcode	Reserved	source	dest
Imm	000100	00	∼rS	rD

```
POP <rD>[, rS]
```

rD = memory[rS]
rS = rS + 2
If no rS is specified, rS defaults to r7 Does not update flags

2.5 MOV

Mov has 2 different encodings depending whether the immediate (if any) fits into 1 byte.
Neither encoding updates the flags.

2.5.1 general MOV encoding

Immediate	opcode	Reserved	source	dest
Imm	001101	00	rS	rD

```
MOV rD, <rS or immediate>
```

rD = op2
This encoding is used for register to register MOVs or when the immediate value will not fit in one byte.

2.5.2 special byte MOV

Reserved	opcode	data
0	000101 + rD	byte immediate

```
MOV rD, <byte immediate>
```

rD = immediate
This encoding is only used when the immediate will fit in 1 byte

2.6 AND

Immediate	opcode	Reserved	source	dest
Imm	001110	00	rS	rD

```
AND rD, <rS or immediate>
```

rD = rD AND op2
This instruction updates the flags, and will reset the overflow and carry flags.

2.7 OR

Immediate	opcode	Reserved	source	dest
Imm	001111	00	rS	rD

```
OR rD, <rS or immediate>
```

rD = rD OR op2
This instruction updates the flags, and will reset the overflow and carry flags.

2.8 XOR

Immediate	opcode	Reserved	source	dest
Imm	010000	00	rS	rD

```
XOR rD, <rS or immediate>
```

rD = rD XOR op2
This instruction updates the flags, and will reset the overflow and carry flags.

2.9 NOT

Immediate	opcode	Reserved	source	dest
Imm	010001	00	000	rD

```
NOT <rD>
```

rD = !rD (bitwise NOT)
This instruction updates the flags, and will reset the overflow and carry flags.

2.10 NEG

Immediate	opcode	Reserved	source	dest
Imm	010010	00	000	rD

```
NEG <rD>
```

rD = 0-rD (signed negation)
This instruction updates the flags.

2.11 LD

Immediate	opcode	byte	displacement	address	data
imm	010011	byte	disp	rS	rD

```
LD{.b} rD, [rS]
LD{.b} rD, [immediate]
LD{.b} rD, [rS+immediate]
```

The load instruction loads a word (or byte) of data from the address specified in the brackets into register rD. The byte flag in the instruction encoding is set when the ".b" suffix is present and indicates a byte load. The displacement flag is set when the third instruction form is used and indicates that rS must be added to the displacement when generating the address. The displacement flag should only be set if the immediate flag is also set. If the displacement flag is set and the immediate flag is not, the behavior is undefined. **Important: All word accesses must be word aligned. Failure to ensure this will result in undefined behavior.**

2.12 ST

Immediate	opcode	byte	displacement	address	data
imm	010100	byte	disp	rS	rD

```
ST{.b} [rS], rD
ST{.b} [immediate], rD
ST{.b} [rS+immediate], rD
```

The store instruction stores the contents of rD into the address specified inside the brackets. The byte flag in the instruction is set when the ".b" suffix is present, and so the processor will only store the least significant 8 bits into the specified address. Similarly to the LD instruction, the disp flag indicates that the processor should add rS and the following immediate value before using the result as the address. **Important: All word accesses must be word aligned. Failure to ensure this will result in undefined behavior.**

2.13 CMP

Immediate	opcode	Reserved	source	dest
Imm	010101	00	rS	rD

```
CMP rD, <rS or immediate>
```

The instruction sets the flags exactly like the SUB instruction, however it does not store the result back to rD.

2.14 JMP

Immediate	opcode	Reserved	condition code	dest
Imm	010110	0	cc	rD

```
JMP.CC <rD or immediate>
```

If the condition code evaluates to True, JMP sets the instruction pointer to the address specified. **Important: This address must be word aligned**

2.15 CALL

Immediate	opcode	Reserved	condition code	dest
Imm	010111	0	cc	rD

```
CALL.CC <rD or immediate>
```

If the condition evaluates to true, CALL saves the instruction pointer in the link register (LR), before setting it to the address specified. **Important: This address must be word aligned**

2.16 RET

Reserved	Opcode	Reserved
0	011000	00000000

```
RET
```

Sets the instruction pointer to the link register(LR).

2.17 SHL

Immediate	opcode	Reserved	source	dest
Imm	011001	00	rS	rD

```
SHL rD, <rS or immediate>
```

Logical left shift rD by op2 and store the result in rD.
Update flags and clear V.

2.18 SHR

Immediate	opcode	Reserved	source	dest
Imm	011010	00	rS	rD

```
SHR rD, <rS or immediate>
```

Logical right shift rD by op2 and store the result in rD.
Update flags and clear V.

2.19 ROL

Immediate	opcode	Reserved	source	dest
Imm	011011	00	rS	rD

```
ROL rD, <rS or immediate>
```

Rotates the bits in rD left by the specified number in op2.
Updates flags and clears carry and overflow.

2.20 RCL

Immediate	opcode	Reserved	source	dest
Imm	011100	00	rS	rD

```
RCL rD, <rS or immediate>
```

Rotates the bits in rD, plus the carry bit left by the value in op2.
Updates flags and clears overflow.

2.21 LDCP

Immediate	opcode	Reserved	Coprocessor Register	dest
0	011101	0	cr	rD

```
LDCP <rD>, <cr>
```

Loads rD with the contents of cr.

2.22 ADC

Immediate	opcode	Reserved	source	dest
Imm	011111	00	rS	rD

```
ADC rD, <rS or immediate>
```

rD = rD + op2 + Carry flag
Updates flags including overflow

2.23 SBB

Immediate	opcode	Reserved	source	dest
Imm	100000	00	rS	rD

```
SBB rD, <rS or immediate>
```

rD = rD - op2 - Carry flag
Updates flags including overflow

2.24 SET

Immediate	opcode	Reserved	condition code	dest
Imm	100001	0	cc	rD

```
SET.CC <rD>
```

If the condition code evaluates to true, sets rD to 1, otherwise sets rD to 0.

2.25 TEST

Immediate	opcode	Reserved	source	dest
Imm	100010	00	rS	rD

```
TEST rD, <rS or immediate>
```

Sets the flags upon the result of the bitwise and of rD and op2. Does not modify the value of rD.

2.26 PUSHLR

Reserved	Opcode	Reserved
0	100011	00000000

```
PUSHLR
```

Pushes the link register onto the stack. This instruction should be used at the beginning of a subroutine if the subroutine executes the `CALL` instruction anywhere in its body. To return after executing this instruction, execute

```
POP r1
JMP.AL r1
```

Note: r1 can be replaced by any GP register

2.27 SAR

Immediate	opcode	Reserved	source	dest
Imm	100100	00	rS	rD

```
SAR rD, <rS or immediate>
```

Shifts the bits in `rD` left by op2 similarly to `SHR`, except the bits shifted in on the left are the sign bit (bit 15) of `rD`. Useful for signed division by powers of 2.

3 Condition Codes

In the `JMP` variety of instructions (`JMP`, `CALL`, `SET`), the condition code field specifies that the instruction should only execute if the expression corresponding to the condition code evaluates to true. The conditions are as follows:

Mnemonic	Encoding	Expression	Meaning
NV	0000	False	Never execute
EQ	0001	Z set	Equal
NE	0010	Z clear	Not equal
OS	0011	V set	Overflow
OC	0100	V clear	No overflow
HI	0101	C set and Z clear	Unsigned greater than
LS	0110	C clear and Z set	Unsigned less than or equal to
P	0111	S clear	Positive
N	1000	S set	Negative
CS	1001	C set	Carry set
CC	1010	C clear	Carry clear
GE	1011	S = V	Signed greater than or equal to
G	1100	S = V and Z clear	Signed greater than
LE	1101	Z set or S $\neq$ V	Signed less than or equal to
L	1110	S $\neq$ V	Signed less than
AL	1111	True	Always execute

The Flags register is as follows:

Bit	Char	Name	Description
0	Z	Zero Flag	Set when the output of the ALU is 0
1	C	Carry Flag	Set when there was a carry out of the MSB in the ALU
2	S	Sign Flag	Set when the MSB of the ALU is 1
3	V	oVerflow Flag	Set when there was a carry out of ALU bit 14 and not one from ALU bit 15

4 Coprocessor Registers

Coprocessor registers are extra registers that *might* be attached to a coprocessor, but may also be extra configuration registers exposed by the cpu. A table is listed below:

Register	alternate names	description
cr0	flags	The processor's condition flags. flags[15:4] = 0 and flags[3:0] = VSCZ
cr1-15	N/A	Unimplemented

5 Peripherals

The D16 processor uses memory mapping to access its peripherals. Currently, the memory mapped region starts at address 0xFF00 and ends at address 0xFFFF. However in subsequent revisions, this may be expanded to the range 0xFC00-0xFFFF, so the programmer is encouraged not to make use of the top 1K of the address space. All peripheral registers will specify their size, either 16 bit or 8 bit. **Important: All registers must be accessed at their stated size. It is illegal to access a 16 bit register as 2 8 bit parts or 2 8 bit registers as a 16 bit register.**

5.1 Peripheral registers

Name	Size	Address	Description
IO_LED_DATA	8	0xFF00	LED Data
IO_UART_DATA	8	0xFF02	UART data
IO_UART_STATUS	8	0xFF03	UART status
IO_UART_BAUD	16	0xFF04	UART Baud rate divisor
IO_TIMER_DATA	16	0xFF06	Timer Data
IO_SOUND_DATA	16	0xFF08	Sound Data
DMA_CONTROL	16	0xFF0A	DMA Control
DMA_LOCAL_ADDRESS	16	0xFF0C	DMA Local Start Address
DMA_REMOTE_ADDRESS	16	0xFF0E	DMA Remote Start Address
DMA_COUNT_ADDRESS	16	0xFF10	DMA Byte Count

5.2 LEDs

5.2.1 IO_LED_DATA

Bits	Type	Description
0-7	W	Output data to LEDs
0-7	R	Read current LED data

5.3 UART

The UART is set up as a pair of 8-entry FIFO queues that feed the Tx and Rx circuitry.

5.3.1 IO_UART_DATA

Bits	Type	Description
0-7	W	Data sent to UART Tx
0-7	R	Data from UART Rx

5.3.2 IO_UART_STATUS

Bits	Type	Description
0	R	Tx FIFO Space free
1	R	Tx FIFO Empty
2	R	Rx Data ready
3	R	Rx FIFO Overrun
4-7	X	Reserved

5.3.3 IO_UART_BAUD

Bits	Type	Description
0-15	W	Baud rate divisor

The baud rate divisor is calculated by

$$divisor = \frac{f_{osc}}{4 * baudrate} \tag{1}$$

Upon reset, the baud rate divisor is set for 115200 baud at 50MHz.

5.4 Timer

5.4.1 IO_TIMER_DATA

Bits	Type	Description
0-15	R/W	Current Timer Count

5.4.2 Usage

The timer counts down from the value set in `IO_TIMER_DATA` to 0 in 1 ms decrements. For the recommended use of waiting for a specified time period, load data into `IO_TIMER_DATA` and poll the register until it returns 0.

5.5 Sound

5.5.1 IO_SOUND_DATA

Bits	Type	Description
0-13	W	Frequency Divisor
14-15	W	Channel selector

5.5.2 Usage

The sound controller contains 4 independant square wave voices with fixed 50% duty cycle and variable frequency. To program a voice's frequency divisor, write the divisor ORed with the voice number shifted 14 bits to the left Ex:

```
mov r0, voice
shl r0, 14
or r0, divisor
st [0xff08], r0
```

The counter for each voice runs at the cpu master clock frequency / 64. The divisor necessary to output a given frequency can be calculated as follows:

$$D = \frac{f_{clk}}{64 * f_{desired}}$$

5.6 DMA

5.6.1 DMA_CONTROL

Bits	Type	Description
0-2	W	DMA type & Start DMA
0-2	R	DMA status (in progress)
3-15	R/W	Reserved, write as 0

5.6.2 DMA_LOCAL_ADDRESS

Bits	Type	Description
0-15	W	Local ram address of data

5.6.3 DMA_REMOTE_ADDRESS

Bits	Type	Description
0-15	W	Upper 16 bits of external DRAM address

5.6.4 DMA_COUNT_ADDRESS

Bits	Type	Description
0-15	W	Number of bytes to transfer

5.6.5 DMA Types

Bits 0-2	Name	Function
000	DMA_WRITE	Initiate write to external DRAM
001	DMA_READ	Initiate read from external DRAM
010-111	RESERVED	Reserved

5.6.6 Usage

The DMA controller allows access to the external DRAM availible to the D16 by asynchronously transferring memory to or from the external DRAM. To use the DMA controller, it is recommended to write to all DMA registers, however the Local address and Remote address registers will contain the address immediately after the final address accessed after a DMA transfer. The recommended usage is to (in this order) program the number of **bytes** to transfer in the DMA Count register, the upper 16 bits of the external ram addres in the DMA Remote register, the word-aligned local address in the DMA Local register, and finally the DMA type in the DMA Control register. Upon writing to the DMA_CONTROL register, the DMA engine will begin transferring data.