The Turtle Assembly Language

Nathaniel Graff

Contents

1 Introduction

This document describes the instruction set architecture and assembly language of the Turtle microcontroller.

2 Registers

2.1 Special Purpose Registers

The Turtle microprocessor has two special purpose registers, the program counter and stack pointer.

2.1.1 pc — Program Counter

The Program Counter (pc) stores the 10-bit memory address of the instruction currently being executed. At initialization and after CPU reset, the pc is set to 0x000. After every execute cycle of the CPU, the program counter is incremented.

The value of the pc can be immediately set with the jmp and call instructions, set to the top value on the stack with the ret instruction, and conditionally set with the jc, jnc, jz, and jnz instructions.

2.1.2 sp — Stack Pointer

The Stack Pointer (sp) stores the 10-bit memory address of the top of the stack. At initialization and after CPU reset, the sp is set to 0x3FF. When values are pushed onto the stack, the sp is decremented, and when values are popped off of the stack, the value is incremented. The sp always holds the address of the next-available space to store a value on the stack, therefore sp + 1 is the memory address of the last value pushed onto the stack.

2.2 General Purpose Registers

The Turtle microcontroller has 8 general purpose 16-bit registers, named r0, r1, r2, ... r7. At CPU initialization and reset, the general purpose registers are set to 0x0000. The high and low byte of each register can be independently set from immediate value using the mvh and mvl instructions, and the value in any register can be cleared (set to 0x0000) using the clr instruction.

Arithmetic operations on registers can only be performed register-to-register, unlike other architectures which often support register-immediate arithmetic operations.

3 Flags

The Turtle microprocessor has two flags, a Carry Flag (c) and Zero Flag (z).

4 Instructions

The Turtle microprocessor uses a fixed-width, 16-bit instruction word. The first 5 bits of each word is reserved for the opcode, leading to a total of 32 available instructions.

4.1 Operations with No Arguments

4.1.1 nop — No Operation

0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-
Opcode					Ignored										

Description No operation is performed
Arguments none
Side effects none

Example Assembly:

```
nop
```

4.1.2 ret — Return from Function

0	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-
Opcode					Ignored										

Description Execution returns from function call by loading the program counter with the value located in memory at the top of the stack (sp + 1).
Arguments none
Side effects The stack pointer is incremented.

Example Assembly:

```
      call myfunc
done: jmp done

myfunc: add r0, r1
        ret
```

4.2 Register Operations

4.2.1 push — Push Register to Stack

1	0	0	0	0	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The value in the argument register is pushed onto the stack.
Arguments The register to push onto the stack
Side effects The stack pointer is decremented.

Example Assembly:

```
mvh  r0, 0x55
mvh  r0, 0xAA
push r0       # 0x55AA is pushed onto the stack
```

4.2.2 pop — Pop from Stack to Register

1	0	0	0	1	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The value at the top of the stack (located at sp + 1) is loaded into the argument register.
Arguments The register to store the value from the stack
Side effects The stack pointer is incremented.

Example Assembly:

```
pop r0
```

4.2.3 in — Store Value from CPU Port to Register

0	0	1	1	0	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description A value is read from the input port into the argument register.
Arguments The register to store the value from the input port
Side effects none

Example Assembly:

```
in r0
```

4.2.4 out — Output Value from Register to CPU Port

0	0	1	1	1	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The value from the argument register is output to the output port.
Arguments The register to output
Side effects none

Example Assembly:

```
mvh r0, 0x55
mvl r0, 0xAA
out r0       # 0x55AA is output from the CPU
```

4.2.5 clr — Clear Register

1	1	0	1	0	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The argument register is set to 0x0000.
Arguments The register to clear
Side effects none

Example Assembly:

```
clr r0
```

4.2.6 lsp (Register) — Load Stack Pointer from Register

1	1	1	0	0	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The stack pointer is set with the bottom 10 bits of the argument register.
Arguments The register to set the stack pointer to
Side effects none

Example Assembly:

```
clr r0
mvl r0, 0xAA
lsp r0       # the stack pointer is set to 0x0AA
```

4.2.7 rsp — Load Stack Pointer into Register

1	1	1	0	1	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The value of the stack pointer is loaded into the argument register.
Arguments The register to store the value of the stack pointer
Side effects none

Example Assembly:

```
rsp r0
```

4.2.8 not — Bitwise Not

1	0	1	1	0	a	a	a	-	-	-	-	-	-	-	-
Opcode					Register			Ignored							

Description The bits of the argument register are inverted and stored back in that register.
Arguments The register to apply the operation to
Side effects If the result is zero, the zero flag is set. The carry flag is cleared.

Example Assembly:

```
clr r0
mvl r0, 0xAA
not r0       # r0 is now 0xFF55
```

4.3 Register-Register Operations

4.3.1 mv — Move Value From Register to Register

0	0	0	0	1	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The value in register B is stored in register A.
Arguments The destination register A and the source register B
Side effects none

Example Assembly:

```
clr r1
mvl r1, 0x55
mv r0, r1    # r0 is now 0x0055
```

4.3.2 add — Add

1	0	0	1	0	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The value in register B is added to the value in register A and stored in register A.
Arguments The two registers to add
Side effects If the addition operation overflows, the carry flag is set. If the result is zero, the zero flag is set.

Example Assembly:

```
add r0, r1
```

4.3.3 sub — Subtract

1	0	0	1	1	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The value in register B is subtracted from the value in register A and the result is stored in register A.
Arguments The two registers to subtract
Side effects If the result is zero, the zero flag is set, if the result underflows, the carry flag is set.

Example Assembly:

```
sub r0, r1
```

4.3.4 cmp — Compare

0	1	0	0	0	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The value in register A is compared to the value in register B. The values in both registers are interpereted as 2's complement.
Arguments The two registers to compare
Side effects If the two registers are equal, the zero flag is set. If register B is greater than register A, the carry flag is set.

Example Assembly:

```
cmp r0, r1
```

4.3.5 and — Bitwise And

1	0	1	0	0	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The values in the two argument registers are bitwise and-ed together and the result stored in register A.
Arguments The two registers to and together
Side effects If the result is zero, the zero flag is set. The carry flag is cleared.

Example Assembly:

```
and r0, r1
```

4.3.6 or — Bitwise Inclusive Or

1	0	1	0	1	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The values in the two argument registers are bitwise inclusive or-ed together and the result stored in register A.

Arguments The two registers to inclusive or together

Side effects If the result is zero, the zero flag is set. The carry flag is cleared.

Example Assembly:

```
or r0, r1
```

4.3.7 xor — Bitwise Exclusive Or

1	0	1	1	1	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The values in the two argument registers are bitwise exclusive or-ed together and the result stored in register A.

Arguments The two registers to exclusive or together

Side effects If the result is zero, the zero flag is set. The carry flag is cleared.

Example Assembly:

```
xor r0, r1
```

4.3.8 shr — Bitwise Shift Right

1	1	0	0	0	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Reg B			Ignored				

Description The value in register A is shifted to the right a number of bits corresponding to the value of register B. Zeros are shifted in on the left.

Arguments Register A is the value to shift and holds the result of the operation. Register B is the number of bits to shift

Side effects If the result is zero, the zero flag is set. The carry flag is cleared.

Example Assembly:

```
shr r0, r1
```

4.3.9 shl — Bitwise Shift Left

1	1	0	0	1	a	a	a	b	b	b	-	-	-	-	-
Opcode					Reg A			Register B			Ignored				

Description The value in register A is shifted to the left a number of bits corresponding to the value of register B. Zeros are shifted in on the right.

Arguments Register A is the value to shift and holds the result of the operation. Register B is the number of bits to shift

Side effects If the result is zero, the zero flag is set. The carry flag is cleared.

Example Assembly:

```
shl r0, r1
```

4.4 Register-Immediate Operations

4.4.1 mvh — Move High Byte into Register

0	0	0	1	0	a	a	a	i	i	i	i	i	i	i	i
Opcode					Register			Immediate							

Description The 8-bit immediate value is stored in the high byte of the argument register. The low byte of the register remains unchanged.

Arguments The destination register and the immediate value

Side effects none

Example Assembly:

```
mvh r0, 0x55
```

4.4.2 mvl — Move Low Byte into Register

0	0	0	1	1	a	a	a	i	i	i	i	i	i	i	i
Opcode					Register			Immediate							

Description The 8-bit immediate value is stored in the low byte of the argument register. The high byte of the register remains unchanged.

Arguments The destination register and the immediate value

Side effects none

Example Assembly:

```
mvl r0, 0xAA
```

4.5 Memory Load and Store Operations

4.5.1 ld (Register Offset) — Load Value from Memory

1	1	1	1	0	a	a	a	b	b	b	o	o	o	o	o
Opcode					Reg A			Reg B			Offset				

Description A value is loaded from memory and stored in register A. The memory address is calculated by taking the bottom 10 bits of register B and adding the immediate value, interpereted as 2's complement.

Arguments The destination register A, the address register B, and the 5-bit 2's complement immediate value offset

Side effects none

Example Assembly:

```
clr r1
mvl r1, 0x10
ld r0, [r1]   # address 0x010
ld r0, [r1+5] # address 0x015
ld r0, [r1-5] # address 0x00B
```

4.5.2 st (Register Offset) — Store Value into Memory

1	1	1	1	1	a	a	a	b	b	b	o	o	o	o	o
Opcode					Reg A			Reg B			Offset				

Description The value in register A is stored in memory. The memory address is calculated by taking the bottom 10 bits of register B and adding the immediate value, interpereted as 2's complement.

Arguments The source register A, the address register B, and the 5-bit 2's complement immediate value offset

Side effects The value at a location in RAM is changed.

Example Assembly:

```
clr r1
mvl r1, 0x10
st r0, [r1]   # address 0x010
st r0, [r1+5] # address 0x015
st r0, [r1-5] # address 0x00B
```

4.5.3 ld (Stack Offset) — Load Value from Memory

0	0	1	0	0	a	a	a	o	o	o	o	o	o	o	o
Opcode					Register			Offset							

Description A value is loaded from memory and stored in register A. The memory address is calculated by taking the stack pointer and adding the immediate value, interpereted as 2's complement.

Arguments The destination register A and the 8-bit 2's complement immediate value offset

Side effects none

Example Assembly:

```
lsp 0x050
ld r0, [sp]    # address 0x050
ld r0, [sp+20] # address 0x064
ld r0, [sp-20] # address 0x03C
```

4.5.4 st (Stack Offset) — Store Value into Memory

0	0	1	0	1	a	a	a	o	o	o	o	o	o	o	o
Opcode					Register			Offset							

Description The value in register A is stored in memory. The memory address is calculated by taking the stack pointer and adding the immediate value, interpereted as 2's complement.

Arguments The source register A and the 8-bit 2's complement immediate value offset

Side effects The value at a location in RAM is changed.

Example Assembly:

```
lsp 0x050
st r0, [sp]    # address 0x050
st r0, [sp+20] # address 0x064
st r0, [sp-20] # address 0x03C
```

4.6 Address Operations

4.6.1 call — Call Function

0	1	1	1	0	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description The value (pc + 1) is pushed onto the stack, and the program counter is set to the argument address.

Arguments The address to set the program counter to

Side effects The stack pointer is decremented.

Example Assembly:

```
      call myfunc
done: jmp done

myfunc: add r0, r1
        ret
```

4.6.2 jmp — Unconditional Jump

0	1	0	0	1	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description The program counter is set to the argument address.
Arguments The address to set the program counter to
Side effects none

Example Assembly:

```
jmp mytag
```

4.6.3 jc — Jump if Carry Flag Set

0	1	0	1	0	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description If the carry flag is set, the program counter is set to the argument address.
Arguments The address to set the program counter to
Side effects none

Example Assembly:

```
jc mytag
```

4.6.4 jnc — Jump if Carry Flag Not Set

0	1	0	1	1	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description If the carry flag is not set, the program counter is set to the argument address.
Arguments The address to set the program counter to
Side effects none

Example Assembly:

```
jnc mytag
```

4.6.5 jz — Jump if Zero Flag Set

0	1	1	0	0	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description If the zero flag is set, the program counter is set to the argument address.
Arguments The address to set the program counter to
Side effects none

Example Assembly:

```
jz mytag
```

4.6.6 jnz — Jump if Zero Flag Not Set

0	1	1	0	1	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description If the zero flag is not set, the program counter is set to the argument address.
Arguments The address to set the program counter to
Side effects none

Example Assembly:

```
jnz mytag
```

4.6.7 lsp (Immediate) — Load Stack Pointer

1	1	0	1	1	i	i	i	i	i	i	i	i	i	i	-
Opcode					Address										

Description The stack pointer is set to the argument address
Arguments The address to set the stack pointer to
Side effects none

Example Assembly:

```
lsp 0x3FF
```

5 Assembler Directives

5.1 .org — Set Origin

Description Place all following assembly at the argument address, until another .org directive is reached.
Arguments The ROM address

Example Assembly:

```
.org 0x100
nop         # address 0x100
nop         # address 0x101
```

5.2 .string — Store hardcoded string in ROM

Description Place the argument string into the ROM, followed by a null byte. Escape characters (newline, tab, etc.) are supported.

Arguments A tag to identify the string and a quoted string

Example Assembly:

```
.string mystring "Hello, world!"
```

5.3 .ldtag — Load the Address of a Tag Into a Register

Description A macro which expands to assembly to store the address of a tag into a register. The result is two operations, a mvh and a mvl.

Arguments A destination register and the tag to store the address of

Example Assembly:

```
.ldtag r0, mytag
```

5.4 .resv — Reserve Space in ROM

Description All following assembly is placed after a gap specified by the argument.

Arguments An optional tag, and the size of the reserved space

Example Assembly:

```
.org 0x100
.resv 25
.resv tagname 25
nop              # address 0x132
```