8X-RIPTIDE Assembler

Table of Contents

Preface

The RIPTIDE Assembler is a lightweight command line tool created to assemble programs for the 8X300, 8X305, 8X-RIPTIDE, RIPTIDE-II, and RIPTIDE-III processors. While simple, the assembler includes features (described in the following section) designed to speed the process of developing, testing, debugging, and deploying assembly code for the various supported targets.

Assembler Features

- Warnings and Errors including
  - Syntax checking
  - Code segment overlap detection
  - Branch target out of range detection
- Integrated pre-processor
  - Supports ‘include’-ing other source files
  - Supports constant defining with the EQU declaration
- Supports macros with parameters
- Can generate a variety of output file types
  - MIF, COE for FPGA memory initialization
  - BIN for general use
- Debug mode for detailed output

Command Line Interface

Usage

At least one of the output file flags (-BIN, -COE, -MIF) must be specified. If more than one output file flag is present, an output file is generated for each.

Arguments

`-ASM <source file>`

`-BIN <output binary>`

`-COE <output coe file>`
The output file will have its file extension replaced or set to ".COE" by the assembler. E.g. providing the file name "foo" or "foo.bar" to the -COE flag will result in "foo.COE" being output by the assembler.

`-MIF <output mif file>`
The output file will have its file extension replaced or set to ".MIF" by the assembler. E.g. providing the file name "foo" or "foo.bar" to the -MIF flag will result in "foo.MIF" being output by the assembler.

`-DEBUG`
The -DEBUG flag has two effects:

1. The pre-processor will print the intermediate pre-proccessed files as output to the terminal. This output is then fed to the assembler as usual.
2. The code generator will print the final machine code in a human-readable format to the terminal.

Machine code output example:

Address	MSB	LSB	Mnemonic	Args...1	...2	...3
0	0	8	JMP	INT_RESET	(8)	
7C	58	29	AND	RIV0	1	R11

Line Syntax

```
LABEL     MNEMONIC OPERANDS ; COMMENT
```

- Label – Must begin with an alphabetical character, can contain only letters and numbers, and must have no white-space preceding it on a line.
- Mnemonic – **Must be preceded by white-space**. Must be an instruction ("MOVE", "ADD", "AND", "XOR", "XEC", "NZT", "XMIT", "JMP", "CALL", "RET", "NOP") or assembler directive (“ORG”, “EQU”, “DATA”)
- Operands – Mnemonic-specific parameters
- Comment – Characters following a ‘;’ will be ignored by the assembler. Comments can be on their own line or on an instruction’s line, following the completed instruction syntax.

Additional Line Structure

If a label appears on its own line, it will be stored by the assembler and mapped/attached to the following line. At most, one label can be attached to a line. If multiple labels appear before a mnemonic, only the last one will be valid.

```
LABEL

          MNEMONIC OPERANDS ; COMMENT
```

In the following example, “LABEL” would not be valid for use and only “LABEL2” will be attached to the following mnemonic. Attempts to use “LABEL” will result in an error.

```
LABEL

LABEL2

          MNEMONIC OPERANDS ; COMMENT
```

If a mnemonic appears at the beginning of the line (i.e. 0 indentation), the assembler will treat it as a label, likely resulting in an error.

```
LABEL

MNEMONIC OPERANDS
```

Reserved Labels

The following keywords cannot be used as labels. Placing them at the beginning of a line results in a specific assembler directive. Also see: directives.

- `INCLUDE`
- `MACRO`
- `ENDMACRO`

Instruction Syntax

NOP

`NOP`

Example:

```
NOP
```

MOVE

`MOVE S(R), D`

Operands:

- S – Source register
- R – Rotate amount
- D – Destination register

Example:

```
MOVE R1(2), AUX
```

Notes:

The rotate amount, R, is an optional field.

E.g. “MOVE R1(0), AUX” is equivalent to “MOVE R1, AUX”

`MOVE S, L, D`

Operands:

- S – Source register
- L – Length
- D – Destination register

Example:

```
MOVE RIV3, 3, R6
```

ADD

`ADD S(R), D`

Operands:

- S – Source register
- R – Rotate amount
- D – Destination register

Example:

```
ADD R1 (4), R3
```

Notes:

The rotate amount, R, is an optional field.

E.g. “ADD R1(0), R3” is equivalent to “ADD R1, R3”

`ADD S, L, D`

Operands:

- S – Source register
- L – Length
- D – Destination register

Example:

```
ADD R11, 4, LIV3
```

AND

`AND S(R), D`

Operands:

- S – Source register
- R – Rotate amount
- D – Destination register

Example:

```
AND R1 (4), R3
```

Notes:

The rotate amount, R, is an optional field.

E.g. “AND R1(0), R3” is equivalent to “AND R1, R3”

`AND S, L, D`

Operands:

- S – Source register
- L – Length
- D – Destination register

Example:

```
AND R11, 4, LIV3
```

XOR

`XOR S(R), D`

Operands:

- S – Source register
- R – Rotate amount
- D – Destination register

Example:

```
XOR R1 (4), R3
```

Notes:

The rotate amount, R, is an optional field.

E.g. “XOR R1(0), R3” is equivalent to “XOR R1, R3”

`XOR S, L, D`

Operands:

- S – Source register
- L – Length
- D – Destination register

Example:

```
XOR LIV7, 4, LIV3
```

XEC

`XEC I (S) [N]`

Operands:

- I – Immediate/label
- S – Source register
- N – *error checking*, expected range of values (S), can't be 0

Example:

```
XEC EX_LABEL (R3) [4]
```

Notes:

The expected range, "N", is an optional field that tells the assembler that the expected values in (S) should vary in the range [0..N-1]. This means that the target location should vary in the range [I..I+N-1]. If the target location [I..I+N-1] can extend beyond the range of XEC, the assembler will emit a warning at assemble time.

If no value is provided for "N", the assembler only checks that "I" is in range of XEC.

`XEC I (S, L) [N]`

Operands:

- I – Immediate/label
- S – Source register
- L – Length
- N – *error checking*, expected range of values (S), can't be 0

Example:

```
XEC EX_LABEL (LIV4,3) [4]
```

Notes:

The expected range, "N", is an optional field that tells the assembler that the expected values in (S) should vary in the range [0..N-1]. This means that the target location should vary in the range [I..I+N-1]. If the target location [I..I+N-1] can extend beyond the range of XEC, the assembler will emit a warning at assemble time.

If no value is provided for "N", the assembler only checks that "I" is in range of XEC.

NZT

`NZT S, I`

Operands:

- S – Source register
- I – Immediate/label

Example:

```
NZT R6, EX_LABEL
```

`NZT S, L, I`

Operands:

- S – Source register
- L – Length
- I – Immediate/label

Example:

```
NZT LIV5, 1, EX_LABEL
```

CALL

`CALL I`

Operands:

- I – Immediate/label

Example:

```
CALL EX_LABEL
```

RET

`RET`

Example:

```
RET
```

XMIT

`XMIT I, D`

Operands:

- I – Immediate/label
- D – Destination register

Example:

```
XMIT @027, IVL
```

`XMIT I, D, L`

Operands:

- I – Immediate/label
- D – Destination register
- L – Length

Example:

```
XMIT 03, LIV5, 3
```

JMP

`JMP A`

Operands:

- A – Address (13-bit field)

Example:

```
JMP EX_LABEL
```

Declarations (Pre-Proccessor)

INCLUDE

Copies the specified file contents into the assembly file being pre-proccessed at the location of the INCLUDE declaration.

Ex.

```
INCLUDE “foo.asm”
```

MACRO / ENDMACRO

Declares a code snippet with parameters that can be referenced throughout the program. There must be exactly one ENDMACRO per MACRO, where MACRO marks the beginning of the macro and ENDMACRO marks the end. After pre-processing, references to the macro in the program will be replaced with the parameterized macro.

Ex.

```
MACRO <macroname> <param1>, <param2>, …
        NOP
        MOVE <param1>, 8, <param2>
        NOP
ENDMACRO <macroname>

        ...
        <macroname> <param1>, <param2>
        ...
```

Ex. 2

```
MACRO nothingburger nothing, burger
        NOP
        MOVE nothing, 8, burger
        NOP
ENDMACRO nothingburger

        ...
        nothingburger R1, R2
        ...
```

EQU

Defines an equivalency, where instances of the identifier are replaced with the associated constant during pre-processing. The associated value must be a constant value, meaning constant expressions such as “3 + 2” are **not** valid.

Ex.

```
<identifier> EQU <constant>
```

Ex. 2

```
FOO EQU 3
```

Directives

ORG

Instructs the assembler to place the code following the ORG directive at the specified address in the code memory region.

Ex.

```
        ORG <addr>
```

Ex. 2

```
        ORG $100
```

DATA

Instructs the assembler to place the specified data in the code memory as if it were machine code. A label is typically placed before the data so that they may be easily accessed, but a label is not required for the directive to work.

Ex.

```
DATALABEL
        DATA <data>
```

Ex. 2

```
DATALABEL
        DATA $DEAD
        DATA $BEEF
```

Ex. 3

```
STRLABEL
        DATA ‘a’
        DATA ‘b’
        DATA ‘c’
```

`HIGH

Instructs the assembler to use only the **high** byte of the constant data following the `HIGH directive. Can be used anywhere the programmer can place constant data in the code.

Ex.

```
XMIT `HIGH $HHLL, R1
```

`LOW

Instructs the assembler to use only the **low** byte of the constant data following the `LOW directive. Can be used anywhere the programmer can place constant data in the code.

Ex.

```
XMIT `LOW $HHLL, R1
```