Pseudo Assembly

A Guide to the Design Philosophy,
Syntax, and Idiosyncrasies of the
Two-Ter Assembly Language

Overview

The Two-Ter Assembly Language, which I call Pseudo Assembly (PASM), is a verbose, slightly clunky, self-descriptive English-like language. In essence, I designed it to be a COBOL-like assembly language. I decided to do it this way, more or less, for the self-description and flow. I feel there's a palpable difference in readability between

```
ADC $1027
STR ACC, $01
LOD ACC .VAL
```

and

```
Add with Carry to Absolute $1027
Store Acc at Paged $01
Load Acc with Vector VAL
```

Another motivation for this decision was user-friendliness. I designed the Two-Ter to also be an educational computer, and an accompanying programming language that feels familiar while still introducing the user to the architecture of the computer is, I feel, much less stressful for a beginner.

To account for use outside of the beginner level, I have included some symbols to replace the more verbose words. For example,

```
Add with Carry to Absolute $1027
```

can be reduced down to

```
Add Carry @$1027
```

Overall, PASM is, I feel, a decent beginner and intermediate language, especially at the low level it operates. I hope you get as much ease of use out of it as I do!

Keyword Description and List

Verbs are the start of every command; they dictate what type of operation the Two-Ter will perform.

Verbs

- Add
- Subtract
- Compare
- AND
- OR
- XOR
- Invert
- Load
- Store
- Move
- Jump
- Subroutine
- Return
- Push
- Pop
- Clear
- Increment
- Decrement
- Rotate
- Shift

Nouns are the physical locations on the Two-Ter and include memory addresses and the registers.

Nouns

- Xh
- Xl
- Yh
- Yl
- H
- L
- Acc
- SP
- PC
- Xhl
- Yhl
- HL
- Flags
- Absolute
- Paged
- Indirect
- Immediate

Meta Nouns are user-defined values of various types that are spliced into the program during code generation.

Meta Nouns

- Line (=)
- Label (.)
- Vector (*)
- Constant (+)
- Word (~)
- Data (:)

Modifiers are optional words that affect the operation of specific instructions.

Modifiers

- If (?)
- Not (!)
- Carry

- Borrow
- Negative
- Zero
- Left
- Right

Prepositions

- With
- At
- To
- Into

Prepositions are words that aid in readability, but don’t affect code operation.

Keyword Substitutions

You probably noticed that some keywords have accompanying symbols in parentheses. These are just shorthand for the keywords for if you get tired of using the full word. For example, “Load Xh with Immediate $FF” can be shortened to “Load Xh #$FF.”

Syntax Legend and List

Legend

() : Required Grouping
[] : Optional Grouping
| : Argument 1 OR Argument 2

Meta Statements

Line $hhll – Sets line in program

Label NAME – Declares label as current line in program

Vector NAME $hhll – Declares a position in memory

Constant NAME ($nn | [-] dec) – Declares a constant

Word [NAME] $hhll "STRING" – Stores a string in memory in ASCII

Data [NAME] $hhll ($nn | [-] dec) – Stores a byte of data in memory

NOTE: If named, Word and Data are declared as Vectors. To refer to them in another statement, i.e. Load, one must use "Load < Register > [with] Vector NAME", not "Load < Register > [with] Data NAME."

Examples:

Line $2000

Label MAIN_LOOP

Vector ADDRESS $8320

Constant NEWLINE $0A

Word $7F00 "Two-Ter User Program Manager v0"

Data CURRENT_ITER $20FE 16

Machine Statements

Halt – Halts operation

Clear (Carry | Negative | Zero | Interrupt | Flags) – Clears flag(s)

(Push | Pop) (Xh | Xl | Yh | Yl | H | L | Acc | SP | Xhl | Yhl | HL | PC)

Examples:

Clear Negative

Clear Flags

Push Acc

Pop HL

Program Flow Statements

Jump [If (([Not] (Carry | Negative | Zero)) | Interrupt)]
[to] ($hhll | Label NAME | Vector NAME) – Direct Jump

Subroutine [If (([Not] (Carry | Negative | Zero)) | Interrupt)]
[to] ($hhll | Label NAME | Vector NAME) – Subroutine

Return [If (([Not] (Carry | Negative | Zero)) | Interrupt)] – Return from Subroutine

Examples:

Jump to Label MAIN_LOOP

Jump if Carry Vector ERROR

Subroutine Vector PROMPT

Return if Zero

Arithmetic and Logic Statements

```
( Rotate | Shift ) [ Acc ] ( Left | Right ) – Rotate/Shift Accumulator

Invert [ Acc ] – NOT Accumulator

( Increment | Decrement ) ( [ Acc ] | Xhl | Yhl | HL )

Add [ [ with ] Carry ] [ to ]

        ( Vector NAME | Absolute $hhll | Paged $nn |
        Indirect ( HL | Xhl | Yhl ) |
        Immediate ( $nn | [ - ] dec ) |
        Constant NAME |
        Xh | Xl | Yh | Yl ) – Addition

Subtract [ [ with ] Borrow ] [ to ]

        ( Vector NAME | Absolute $hhll | Paged $nn |
        Indirect ( HL | Xhl | Yhl ) |
        Immediate ( $nn | [ - ] dec ) |
        Constant NAME |
        Xh | Xl | Yh | Yl ) – Subtraction

( Compare | AND | OR | XOR ) [ to ]

        ( Vector NAME | Absolute $hhll | Paged $nn |
        Indirect ( HL | Xhl | Yhl ) |
        Immediate ( $nn | [ - ] dec ) |
        Constant NAME |
        Xh | Xl | Yh | Yl ) – Comparison and Bitwise Logic Functions
```

Examples:

```
        Add with Carry to Xh

        Add Constant OFFSET

        Subtract Immediate 127

        Subtract with Borrow Yh

        Compare to Vector CHECK_VAL

        AND Xh
```

Load, Store and Move Statements

Load (Xh | Xl | Yh | Yl | H | L | Acc | SP) [with]

(Vector NAME | Absolute $hhll | Paged $nn |
Immediate ($nn | [-] dec) |
Constant NAME) – Load value from memory into specified register

Store (Xh | Xl | Yh | Yl | H | L | Acc | SP) [at]

(Vector NAME | Absolute $hhll | Paged $nn) – Store value from specified register into memory

Move (Xh | Xl | Yh | Yl | H | L | Acc | SP) [to | into]
(Xh | Xl | Yh | Yl | H | L | Acc | SP) – 8 Bit Move

Move (Xhl | Yhl | HL | PC) [to | into] (Xhl | Yhl | HL | PC) – 16 Bit Move

Examples:

Load Acc with Absolute $FF00

Load Xh Immediate -23

Store Acc at Vector CURRENT_ITER

Move SP into Acc

Move Xh to Yh

Move PC to Xhl