User Manual - Pullet_16 Assembler

Group 5

Sean A Wiig, Mayank D Patel, Tsung Wei Wu, Mark McMurtury Jr, Samyuktha M Comandur

Homework 6B

6 December 2018

This user manual will serve as a way of communicating how our assembler works for someone who is unfamiliar with C++.

Helpful links before diving in:

- More About C++
  - How to run C++ programs, make files, etc
  - Vectors
  - Maps
  - ASCII vs binary
  - Bitwise math
  - Labels
  - Hexadecimal
  - Symbols
  - Opcodes
  - Memory Address
  - Memory Addressing

The Pullet_16 Computer

The Pullet16 is a machine with 4096 words of 16 bits each. There is no distinction between program memory and data memory, no virtual memory, etc. This is a single-user system and the one user at a time able to access the machine gets all of it. There is no memory protection; any user can access all of memory. (learn more)

In this user manual you will learn:

- The Pullet_16 Computer
- How To Run The Program
- How To Write Machine Code
- How To Write Source Code and Check for Errors

How To Run The Program

First, we must start up our terminal and locate the directory in which the program is located. To do this we must type "cd" (learn about changing directories) which changes directory to a path defined.

```
$ cd "directory_name"
```

Now to run the program and depending on where your makefile is located:

First, run the makefile (see more)

```
$ make -f "file_name"
```

Second, run the assembler program by feeding it 3 command line arguments. The input file, output file, and the log file is required for execution.(learn about command line arguments)

```
$ ./Aprog "input_filename" "output_filename" "log_filename"
```

Example:

The first line of command compiles the C++ programs (The makefile is already written).

The second line of command runs the assembler program while feeding it the 3 arguments.

1. The first argument is the source code file name (ie. ../../ytestbr.txt), which is located in the main directory of the assignment.
2. The second argument is the output file name (ie. adotouttestbr), which does not include an extension because both the .bin and .txt files will be dealt with in the assignment.
3. The third argument is the log file name (ie. ytestbrlog.txt) which consists of all the logs recorded within running the program.

```
$ make -f ../../makefile
$ ./Aprog ../../ytestbr.txt adotouttestbr ytestbrlog.txt
```

Output:

```
  Line Mem  Mem Loc      Machine             SYM L  Mne A SYM HexNo Comment
   Num  Loc Binary       Code
    0                                          *23 567 9 123 56789 1
    1                                          *ll mmm a sss hhhhh * comment
    2    0  000000000000 1010 0000 0000 1000        LD    A         * load the first addend
    3    1  000000000001 1110 0000 0000 0011        WRT             * write the acc to output
    4    2  000000000010 1100 0000 0000 0101        BR    SKP       * add in the second addend
    5    3  000000000011 1110 0000 0000 0011        WRT             * write the acc to output
    6    4  000000000100 1110 0000 0000 0010        STP   ... .....
    7    5  000000000101 1010 0000 0000 1001 SKP    LD    B         * load the first addend
    8    6  000000000110 1110 0000 0000 0011        WRT             * write the acc to output
    9    7  000000000111 1110 0000 0000 0010        STP   ... .....
   10    8  000000001000 0000 0000 1111 0000        END   ... .....
   11                                           *
   12    8  000000001000 0000 0000 0000 0101 A      HEX       +0005
   13    9  000000001001 0000 0000 0000 1001 B      HEX       +0009

SYMBOL TABLE
    SYM LOC FLAGS
SYM A     8
SYM B     9
SYM SKP   5

MACHINE CODE
enter PrintMachineCode adotouttestbr.bin
   0 000000000000 1010 0000 0000 1000
   1 000000000001 1110 0000 0000 0011
   2 000000000010 1100 0000 0000 0101
   3 000000000011 1110 0000 0000 0011
   4 000000000100 1110 0000 0000 0010
   5 000000000101 1010 0000 0000 1001
   6 000000000110 1110 0000 0000 0011
   7 000000000111 1110 0000 0000 0010
   8 000000001000 0000 0000 0000 0101
   9 000000001001 0000 0000 0000 1001
```

The assembler compiles the source code, creates a symbol table checking for error flags and prints out the Machine Code in the output binary file. (Source Code further explained in page 5)

(See ‘System Manual’ for more explanation on how the program works.)

How To Write Machine Code

First let's examine how the Machine Code is formatted and works. (learn more)

There are two machine-instruction formats. These are the binary patterns that are the machine code that is part of the analog to an a.out file. (learn more)

• Machine Code Format I:
    bits 0-2 opcode
    bit 3 0 value indicates direct addressing, 1 indicates indirect
    bits 4-15 memory address in hexadecimal

• Machine Code Format II:
    bits 0-2 opcode
    bit 3 forced zero
    bits 4-15 function selector code

The complete instruction set for the Pullet16 is as follows.

Format I Mnemonic opcode	Binary opcode	Description
BAN	000	Branch on ACC negative
SUB	001	Subtract contents of memory from ACC
STC	010	Store ACC in memory and then clear ACC
AND	011	And ACC with contents of memory
ADD	100	Add contents of memory to ACC
LD	101	Load ACC from contents of memory
BR	110	Unconditional branch

Format II Mnemonic opcode	Binary opcode	Hex opcode	Description
RD	1110 0000 0000 0001	E001	Read from standard input into ACC
STP	1110 0000 0000 0010	E002	STOP execution
WRT	1110 0000 0000 0011	E003	Write from ACC to standard output

How To Write Source Code and Check For Errors

First the format must be clearly specified and the program format for lines of input is as follows.
If column 1 is an asterisk, the entire line is a comment. Otherwise, the format is for fixed columns:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L	**L**	**L**		**M**	**M**	**M**		**A**		**S**	**S**	**S**		**+/-**	**H**	**H**	**H**	**H**		**C**

bits 0 - 2 (**LLL**): Label. An optional alphanumeric field that must be less than or equal to three characters long. Must begin with an alphabetic character. (learn more).

bits 4 - 6 (**MMM**): Mnemonic opcode. This opcode must be one of the following: BAN, SUB, STC, AND, ADD, LD, BR, RD, STP, WRT or one of the following pseudo-op instructions for the assembler: DS, HEX, ORG, END. (discussed on page 4). (learn more).

bit 8 (**A**): Addressing flag. An asterisk indicates direct addressing, a blank space here indicates indirect addressing. (learn more).

bits 10 - 12 (**SSS**): Symbolic operand. This field follows the same restrictions as the label (Field must be less than or equal to three characters long. Must begin with an alphabetic character.) (learn more)

bits 14 - 18 (**+/-HHHH**): Hex operand. +/- indicates that the first bit of the hex operand must indicate whether the operand is positive or negative, and the other 4 are characters and must be valid hexadecimal digits (numbers 0 - 9, capital letters A - F). (learn more)

bits 20+ : (**C…**): Comments. Any comments are denoted by an asterisk and continue for the remainder of the line, meaning that this field has a variable length, unlike any other field. (learn more)

More Errors

- Addresses must be less than 4096 decimal (Therefore, the hex values must be checked).
- ORG must be bounded between 0 and 4096.
- DS must be bounded between 0 and 4096.
- Labels refer to memory locations and thus cannot be multiply defined.
- Symbols must be defined.
- END statement must be included.
- Any source code not following the format listed above will bound to run into errors.

Example:

Here Pass One had already created the symbol table and flagged the errors then printed it in Pass Two.

1. ‘STR’ is defined more than one time, thus the symbol is multiply defined. (line 1 & 5)
2. ‘68 ’ is not an alphanumeric field that begin with an alphabetic character, thus the symbol is invalid. (line 3)
3. ‘CHK’ and ‘90 ’ was not defined in the label column and was called in the symbol column, thus the symbol is undefined. (line 4 & 6)
4. ‘DIV’ was not part of the mnemonic opcode nor pseudo-op instructions defined in page 4 & 5, thus the mnemonic is invalid. (line 6)
5. Hex value ‘+F00F’ is a decimal value of ‘61,455’ and one cannot DS above 4096 as mentioned in page 5, thus DS Allocation is invalid. (line 12)
6. ‘END’ statement was not found, thus no end statement error was shown. (line 14)

```
PASS TWO
    0                                              *23456789012345678901
    1    0  000000000000 1110 0000 0000 0001 STR    RD    ... .....
***** ERROR -- SYMBOL STR IS MULTIPLY DEFINED

    2    1  000000000001 0100 0000 0000 0110        STC   A1  .....
    3    2  000000000010 0111 0000 0001 1001 68     AND * B1  .....
***** ERROR -- SYMBOL 68  IS INVALID

    4    3  000000000011 0000 1111 0000 1111        BAN   CHK .....
***** ERROR -- SYMBOL CHK IS UNDEFINED

    5    4  000000000100 1011 0000 0000 0110 STR    LD  * A1  .....
***** ERROR -- SYMBOL STR IS MULTIPLY DEFINED

    6    5  000000000101 1100 1100 1100 1100        DIV   90  .....
***** ERROR -- SYMBOL 90  IS UNDEFINED
***** ERROR -- MNEMONIC DIV IS INVALID

    7    6  000000000110 1111 0000 0000 0000 A1     DS        +000A * comment
    8   16  000000010000 1111 0000 0000 0000        DS        +0006 * comment
    9   22  000000010110 0000 1010 0000 1010        HEX       +0A0A
   10   23  000000010111 1111 0010 0000 0010        HEX       +F202
   11   24  000000011000 0001 1110 1111 1111        HEX       -E101
   12   25  000000011001 1100 1100 1100 1100        DS        +F00F * comment
***** ERROR -- DS ALLOCATION +F00F IS INVALID

   13   25  000000011001 0000 0000 0001 0000 B1     HEX       +0010 * comment
   14   26  000000011010 1111 1111 1110 0100 B2     HEX       -001C * comment

***** ERROR -- NO 'END' STATEMENT

SYMBOL TABLE
    SYM LOC FLAGS
SYM 68    2 INVALID
SYM A1    6
SYM B1   25
SYM B2   26
SYM STR   0 MULTIPLY



ERRORS EXIST
NO MACHINE CODE GENERATED
Main: Ending execution
```